

5. DIVIDE AND CONQUER I

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *randomized quicksort*
- ▶ *median and selection*
- ▶ *closest pair of points*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>

Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems (of the same kind).
- Solve (conquer) each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Most common usage.

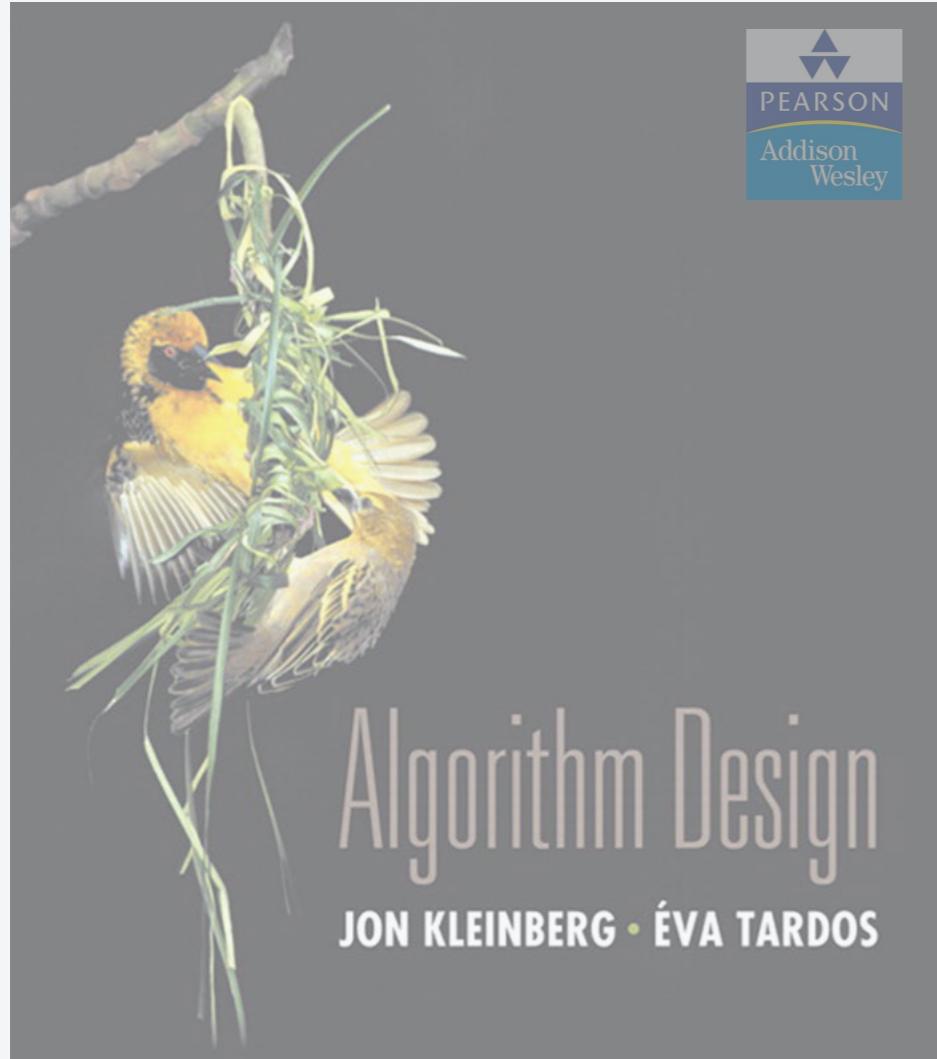
- Divide problem of size n into two subproblems of size $n/2$. $\leftarrow O(n)$ time
- Solve (conquer) two subproblems recursively.
- Combine two solutions into overall solution. $\leftarrow O(n)$ time

Consequence.

- Brute force: $\Theta(n^2)$.
- Divide-and-conquer: $O(n \log n)$.



attributed to Julius Caesar



SECTIONS 5.1–5.2

5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *randomized quicksort*
- ▶ *median and selection*
- ▶ *closest pair of points*

Sorting problem

Problem. Given a list L of n elements from a totally ordered universe, rearrange them in ascending order.

The image shows a digital music player interface. At the top, there is a horizontal scrollable view displaying several album covers. Below this is a table listing 38 songs. The columns are labeled: Row Number, Name, Artist, Time, and Album. The table is sorted by Name. The song "Dancing In The Dark" by Bruce Springsteen is highlighted with a blue selection bar at the bottom. The table data is as follows:

	Name	Artist	Time	Album
12	<input checked="" type="checkbox"/> Let It Be	The Beatles	4:03	Let It Be
13	<input checked="" type="checkbox"/> Take My Breath Away	BERLIN	4:13	Top Gun – Soundtrack
14	<input checked="" type="checkbox"/> Circle Of Friends	Better Than Ezra	3:27	Empire Records
15	<input checked="" type="checkbox"/> Dancing With Myself	Billy Idol	4:43	Don't Stop
16	<input checked="" type="checkbox"/> Rebel Yell	Billy Idol	4:49	Rebel Yell
17	<input checked="" type="checkbox"/> Piano Man	Billy Joel	5:36	Greatest Hits Vol. 1
18	<input checked="" type="checkbox"/> Pressure	Billy Joel	3:16	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
19	<input checked="" type="checkbox"/> The Longest Time	Billy Joel	3:36	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
20	<input checked="" type="checkbox"/> Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
21	<input checked="" type="checkbox"/> Sunday Girl	Blondie	3:15	Atomic: The Very Best Of Blondie
22	<input checked="" type="checkbox"/> Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
23	<input checked="" type="checkbox"/> Dreaming	Blondie	3:06	Atomic: The Very Best Of Blondie
24	<input checked="" type="checkbox"/> Hurricane	Bob Dylan	8:32	Desire
25	<input checked="" type="checkbox"/> The Times They Are A-Changin'	Bob Dylan	3:17	Greatest Hits
26	<input checked="" type="checkbox"/> Livin' On A Prayer	Bon Jovi	4:11	Cross Road
27	<input checked="" type="checkbox"/> Beds Of Roses	Bon Jovi	6:35	Cross Road
28	<input checked="" type="checkbox"/> Runaway	Bon Jovi	3:53	Cross Road
29	<input checked="" type="checkbox"/> Rasputin (Extended Mix)	Boney M	5:50	Greatest Hits
30	<input checked="" type="checkbox"/> Have You Ever Seen The Rain	Bonnie Tyler	4:10	Faster Than The Speed Of Night
31	<input checked="" type="checkbox"/> Total Eclipse Of The Heart	Bonnie Tyler	7:02	Faster Than The Speed Of Night
32	<input checked="" type="checkbox"/> Straight From The Heart	Bonnie Tyler	3:41	Faster Than The Speed Of Night
33	<input checked="" type="checkbox"/> Holding Out For A Hero	Bonnie Tyler	5:49	Meat Loaf And Friends
34	<input checked="" type="checkbox"/> Dancing In The Dark	Bruce Springsteen	4:05	Born In The U.S.A.
35	<input checked="" type="checkbox"/> Thunder Road	Bruce Springsteen	4:51	Born To Run
36	<input checked="" type="checkbox"/> Born To Run	Bruce Springsteen	4:30	Born To Run
37	<input checked="" type="checkbox"/> Jungleland	Bruce Springsteen	9:34	Born To Run
38	<input checked="" type="checkbox"/> Turn! Turn! Turn! (To Everything)	The Byrds	2:57	Forrest Gump: The Soundtrack (Disc 2)

Sorting applications

Obvious applications.

- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

Some problems become easier once elements are sorted.

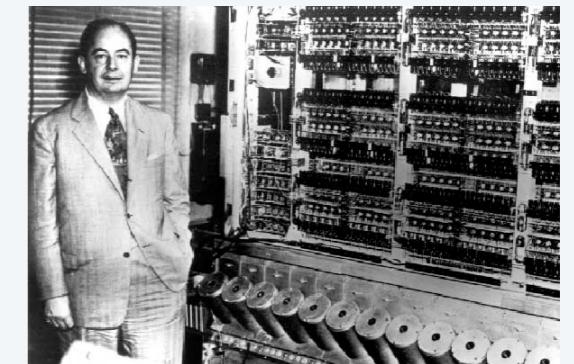
- Identify statistical outliers.
- Binary search in a database.
- Remove duplicates in a mailing list.

Non-obvious applications.

- Convex hull.
- Closest pair of points.
- Interval scheduling / interval partitioning.
- Scheduling to minimize maximum lateness.
- Minimum spanning trees (Kruskal's algorithm).
- ...

Mergesort

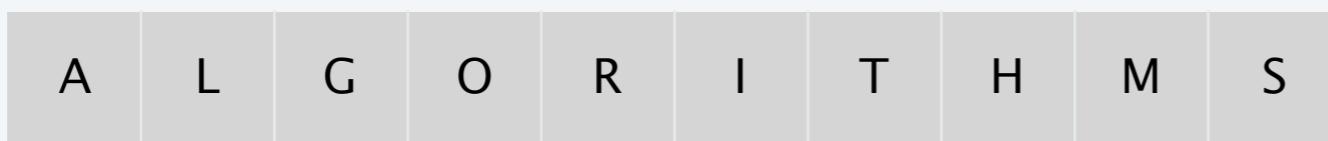
- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.



**First Draft
of a
Report on the
EDVAC**

John von Neumann

input



sort left half



sort right half



merge results



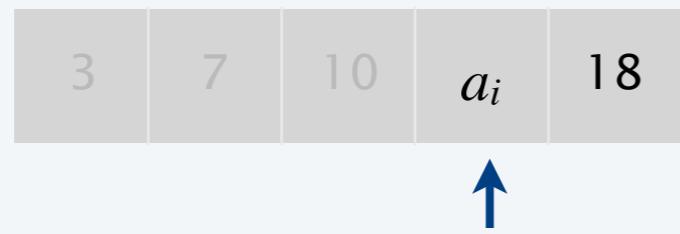
Merging

Goal. Combine two sorted lists A and B into a sorted whole C .

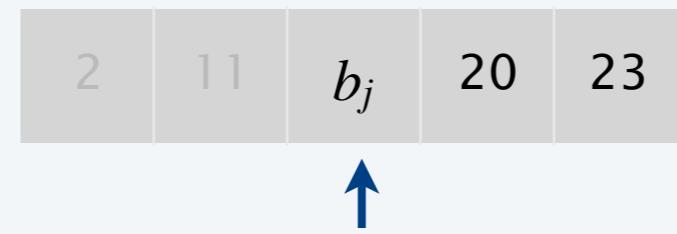


- Scan A and B from left to right.
- Compare a_i and b_j .
- If $a_i \leq b_j$, append a_i to C (no larger than any remaining element in B).
- If $a_i > b_j$, append b_j to C (smaller than every remaining element in A).

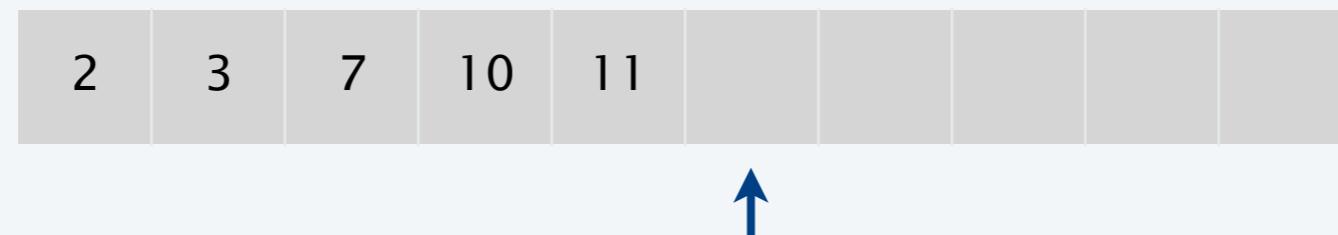
sorted list A



sorted list B



merge to form sorted list C



Mergesort implementation

Input. List L of n elements from a totally ordered universe.

Output. The n elements in ascending order.

MERGE-SORT(L)

IF (list L has one element)

RETURN L .

Divide the list into two halves A and B .

$A \leftarrow \text{MERGE-SORT}(A).$ $\longleftarrow T(n / 2)$

$B \leftarrow \text{MERGE-SORT}(B).$ $\longleftarrow T(n / 2)$

$L \leftarrow \text{MERGE}(A, B).$ $\longleftarrow \Theta(n)$

RETURN L .

A useful recurrence relation

Def. $T(n)$ = max number of compares to mergesort a list of length n .

Recurrence.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & \text{if } n > 1 \end{cases}$$


between $\lfloor n / 2 \rfloor$ and $n - 1$ compares

Solution. $T(n)$ is $O(n \log_2 n)$.

Assorted proofs. We describe several ways to solve this recurrence.

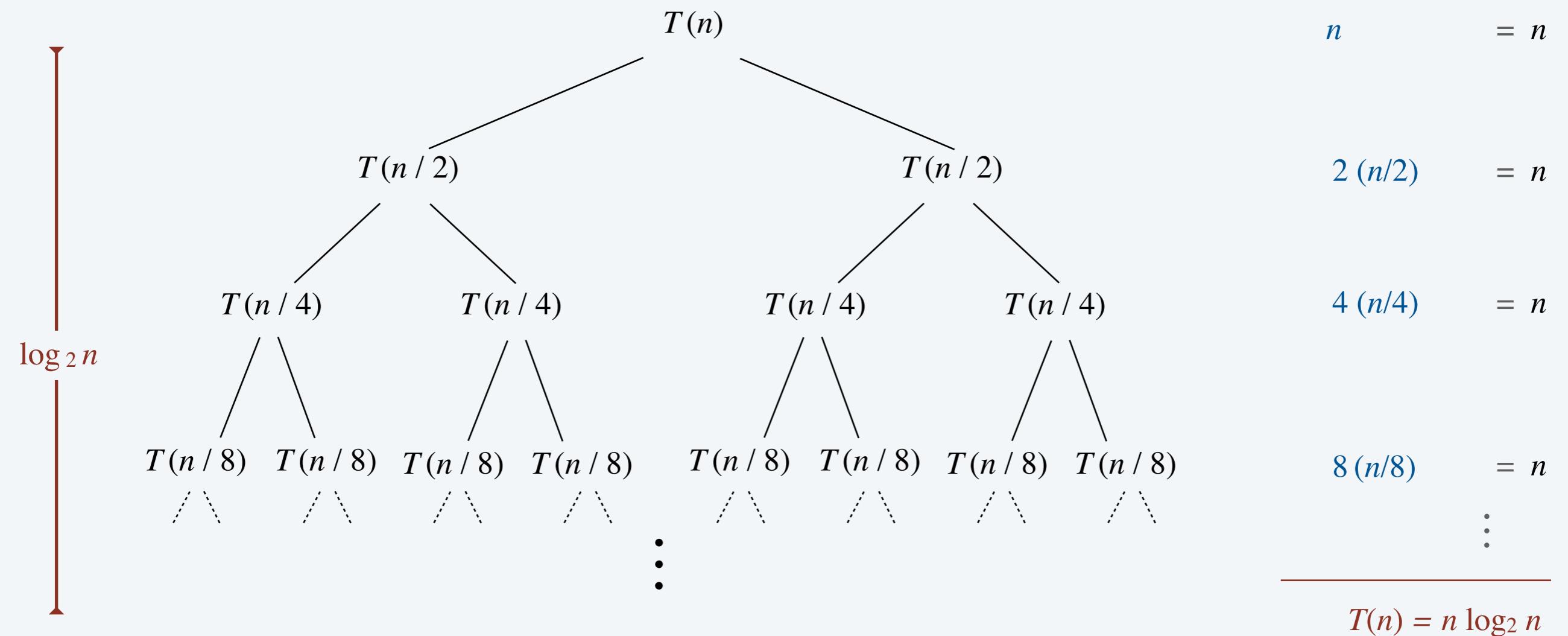
Initially we assume n is a power of 2 and replace \leq with $=$ in the recurrence.

Divide-and-conquer recurrence: recursion tree

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$$

assuming n
is a power of 2



Proof by induction

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$$

assuming n
is a power of 2

Pf. [by induction on n]

- Base case: when $n = 1$, $T(1) = 0 = n \log_2 n$.
- Inductive hypothesis: assume $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

$$T(2n) = 2T(n) + 2n$$

recurrence

inductive hypothesis \rightarrow $= 2n \log_2 n + 2n$

$$\begin{aligned} &= 2n(\log_2(2n) - 1) + 2n \\ &= 2n \log_2(2n). \blacksquare \end{aligned}$$



Which is the exact solution of the following recurrence?

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n - 1 & \text{if } n > 1 \end{cases}$$

 no longer assuming n is a power of 2

- A. $T(n) = n \lfloor \log_2 n \rfloor$
- B. $T(n) = n \lceil \log_2 n \rceil$
- C. $T(n) = n \lfloor \log_2 n \rfloor + 2^{\lceil \log_2 n \rceil} - 1$
- D. $T(n) = n \lceil \log_2 n \rceil - 2^{\lfloor \log_2 n \rfloor} + 1$
- E. Not even Knuth knows.

Analysis of mergesort recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lceil \log_2 n \rceil$.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & \text{if } n > 1 \end{cases}$$

↑
no longer assuming n
is a power of 2

Pf. [by strong induction on n]

- Base case: $n = 1$.
- Define $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$ and note that $n = n_1 + n_2$.
- Induction step: assume true for $1, 2, \dots, n-1$.

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n \\ \text{inductive hypothesis } \rightarrow &\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &\leq n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &= n \lceil \log_2 n_2 \rceil + n \\ &\leq n (\lceil \log_2 n \rceil - 1) + n \quad \leftarrow \text{log}_2 n_2 \leq \lceil \log_2 n \rceil - 1 \\ &= n \lceil \log_2 n \rceil. \blacksquare \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\leq \lceil 2^{\lceil \log_2 n \rceil} / 2 \rceil \\ &= 2^{\lceil \log_2 n \rceil} / 2 \end{aligned}$$

\uparrow
an integer

Digression: sorting lower bound

Challenge. How to prove a lower bound for **all** conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

Cost model. Number of compares.

Q. Realistic model?

A1. Yes. Java, Python, C++, ...

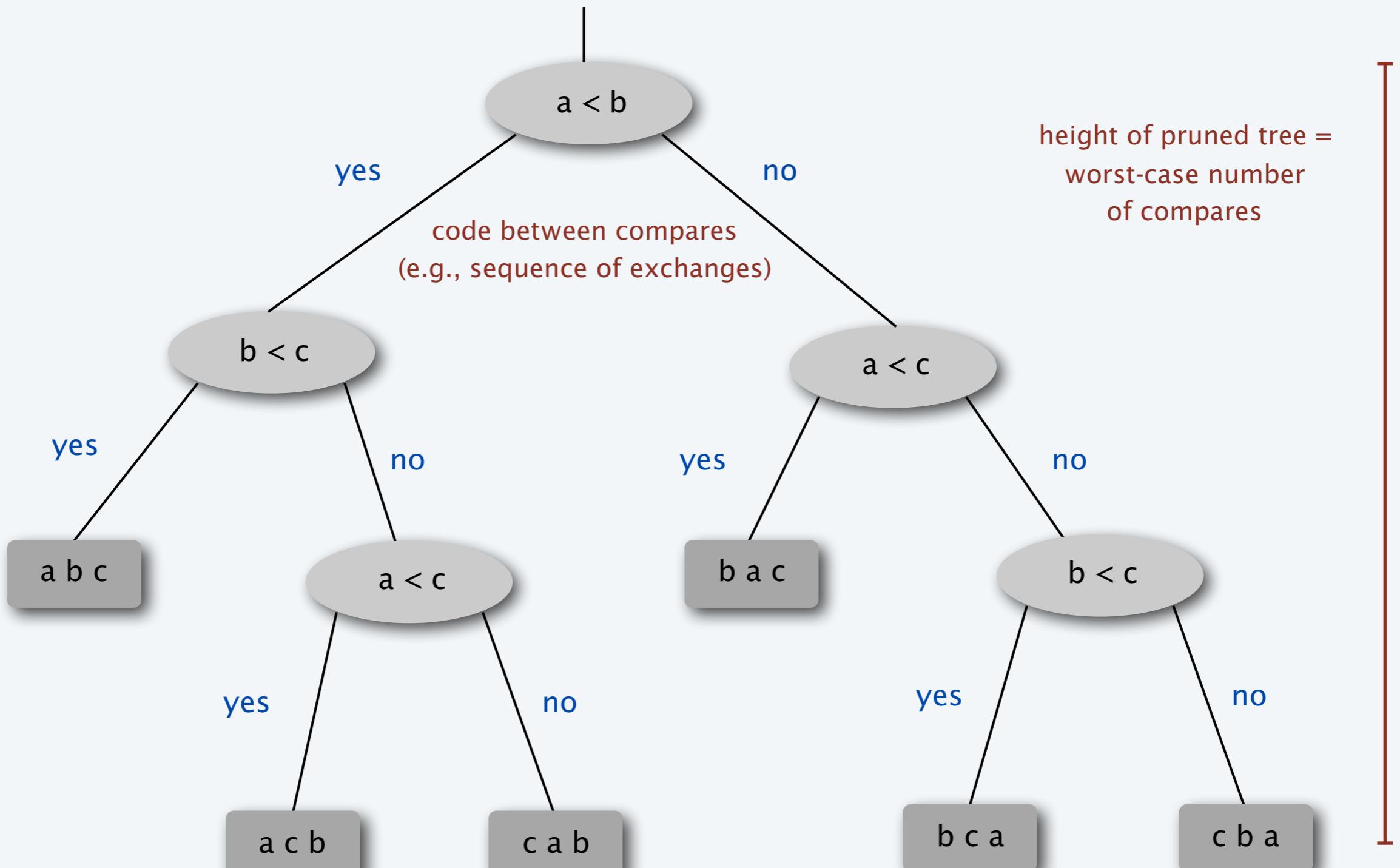
A2. Yes. Mergesort, insertion sort, quicksort, heapsort, ...

A3. No. Bucket sort, radix sorts, ...

sort(*, key=None, reverse=False)

This method sorts the list in place, using only `<` comparisons between items. Exceptions are not suppressed – if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

Comparison tree (for 3 distinct keys a, b, and c)



Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n .
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.
- $n!$ different orderings $\Rightarrow n!$ reachable leaves.



Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n .
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.
- $n!$ different orderings $\Rightarrow n!$ reachable leaves.

$$\begin{aligned} 2^h &\geq \# \text{ leaves} \geq n! \\ \Rightarrow h &\geq \log_2(n!) \\ &\geq n \log_2 n - n / \ln 2 \quad \blacksquare \end{aligned}$$

↑
Stirling's formula



Note. Lower bound can be extended to include randomized algorithms.

SHUFFLING A LINKED LIST

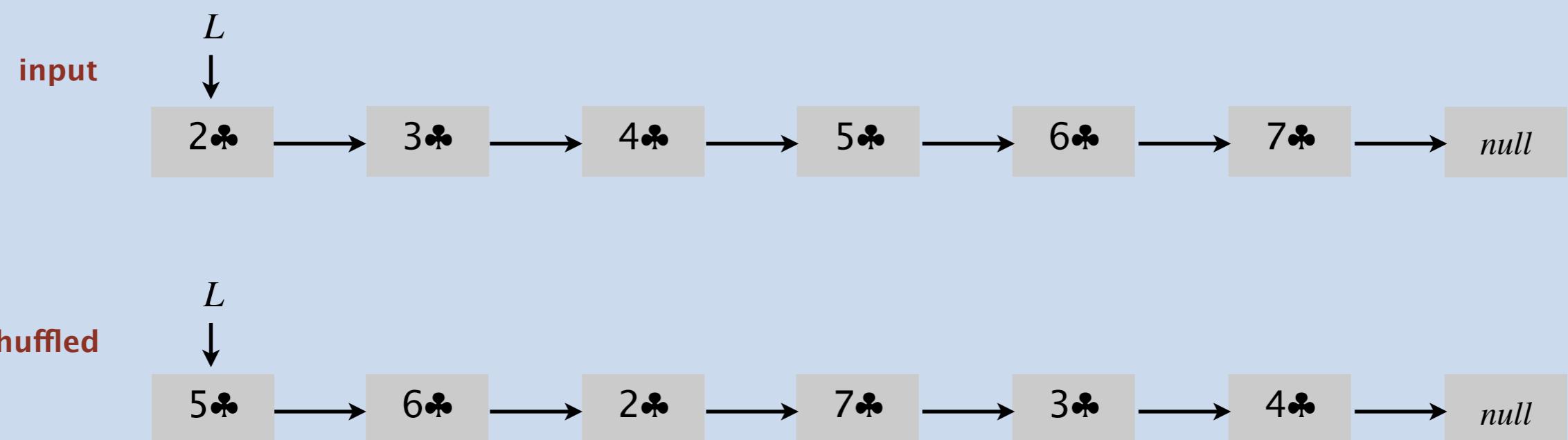


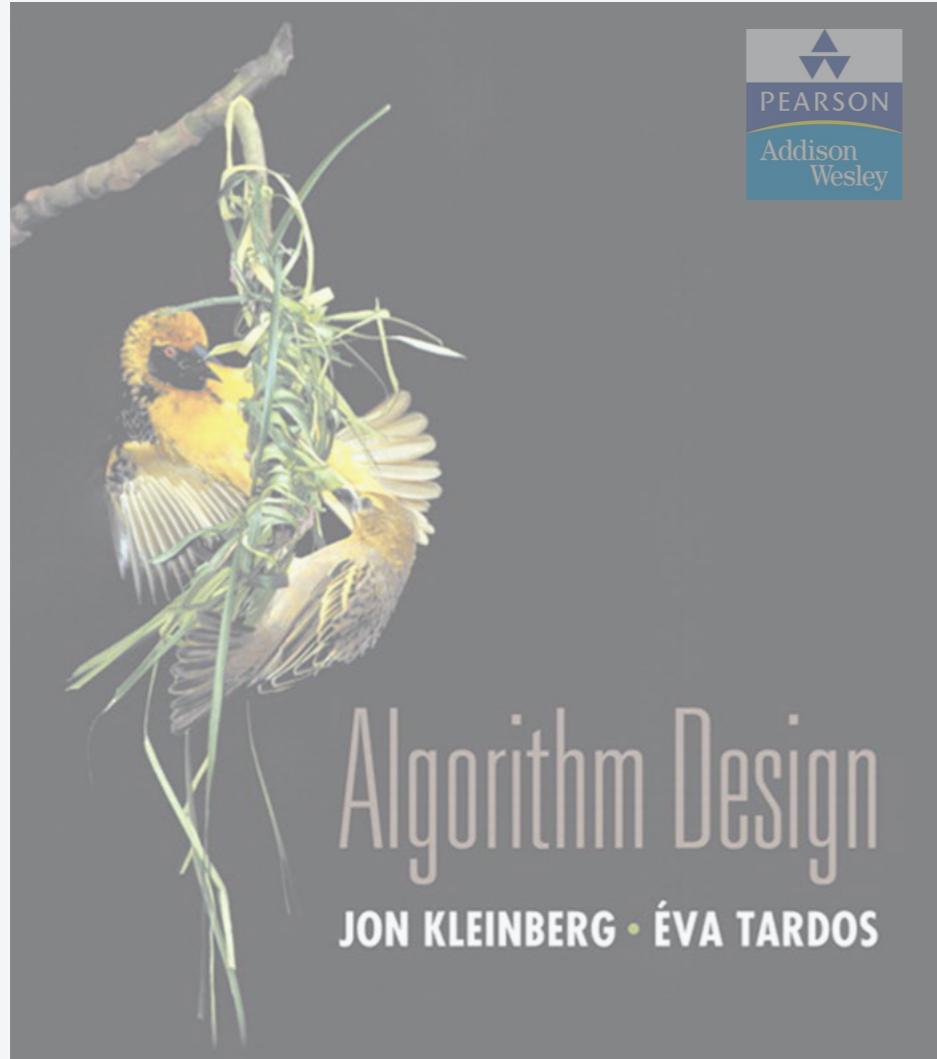
Problem. Given a singly linked list, rearrange its nodes uniformly at random.

Assumption. Access to a perfect random-number generator.

all $n!$ permutations
equally likely

Performance. $O(n \log n)$ time, $O(\log n)$ extra space.





SECTION 5.3

5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *randomized quicksort*
- ▶ *median and selection*
- ▶ *closest pair of points*

Counting inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of **inversions** between two rankings.

- My rank: $1, 2, \dots, n$.
- Your rank: a_1, a_2, \dots, a_n .
- Songs i and j are inverted if $i < j$, but $a_i > a_j$.

	A	B	C	D	E
me	1	2	3	4	5
you	1	3	4	2	5

2 inversions: 3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs.

Counting inversions: applications

- Voting theory.
- Collaborative filtering.
- Measuring the “sortedness” of an array.
- Sensitivity analysis of Google’s ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall’s tau distance).

Rank Aggregation Methods for the Web

Cynthia Dwork*

Ravi Kumar†

Moni Naor‡

D. Sivakumar§

ABSTRACT

We consider the problem of combining ranking results from various sources. In the context of the Web, the main applications include building meta-search engines, combining ranking functions, selecting documents based on multiple criteria, and improving search precision through word associations. We develop a set of techniques for the rank aggregation problem and compare their performance to that of well-known methods. A primary goal of our work is to design rank aggregation techniques that can effectively combat “spam,” a serious problem in Web searches. Experiments show that our methods are simple, efficient, and effective.

Keywords: rank aggregation, ranking functions, meta-search, multi-word queries, spam

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B .
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with $a \in A$ and $b \in B$.
- Return sum of three counts.

input

1	5	4	8	10	2	6	9	3	7
---	---	---	---	----	---	---	---	---	---

count inversions in left half A

1	5	4	8	10
---	---	---	---	----

5-4

count inversions in right half B

2	6	9	3	7
---	---	---	---	---

6-3 9-3 9-7

count inversions (a, b) with $a \in A$ and $b \in B$

1	5	4	8	10
---	---	---	---	----

2	6	9	3	7
---	---	---	---	---

4-2 4-3 5-2 5-3 8-2 8-3 8-6 8-7 10-2 10-3 10-6 10-7 10-9

output $1 + 3 + 13 = 17$

Counting inversions: how to combine two subproblems?

Q. How to count inversions (a, b) with $a \in A$ and $b \in B$?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B .
- For each element $b \in B$,
 - binary search in A to find how elements in A are greater than b .

list A

7	10	18	3	14
---	----	----	---	----

list B

20	23	2	11	16
----	----	---	----	----

sort A

3	7	10	14	18
---	---	----	----	----

sort B

2	11	16	20	23
---	----	----	----	----

binary search to count inversions (a, b) with $a \in A$ and $b \in B$

3	7	10	14	18
---	---	----	----	----

2	11	16	20	23
---	----	----	----	----

5 2 1 0 0

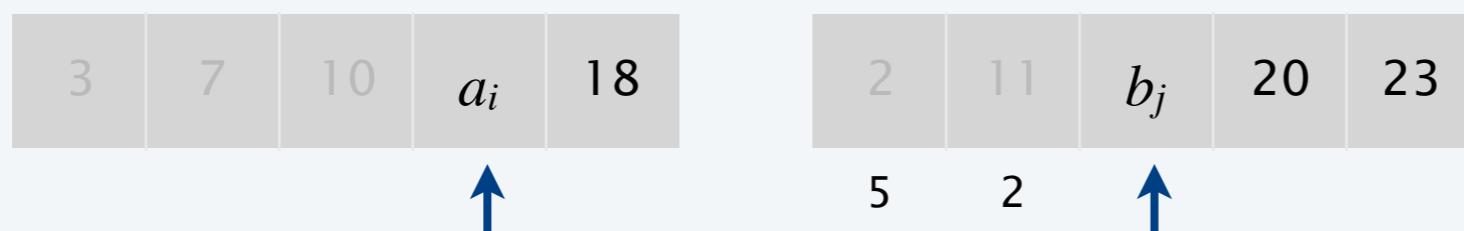
Counting inversions: how to combine two subproblems?

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j .
- If $a_i < b_j$, then a_i is not inverted with any element left in B .
- If $a_i > b_j$, then b_j is inverted with every element left in A .
- Append smaller element to sorted list C .



count inversions (a, b) with $a \in A$ and $b \in B$



merge to form sorted list C



Counting inversions: divide-and-conquer algorithm implementation

Input. List L .

Output. Number of inversions in L and L in sorted order.

SORT-AND-COUNT(L)

IF (list L has one element)

RETURN $(0, L)$.

Divide the list into two halves A and B .

$(r_A, A) \leftarrow \text{SORT-AND-COUNT}(A).$ $\longleftarrow T(n / 2)$

$(r_B, B) \leftarrow \text{SORT-AND-COUNT}(B).$ $\longleftarrow T(n / 2)$

$(r_{AB}, L) \leftarrow \text{MERGE-AND-COUNT}(A, B).$ $\longleftarrow \Theta(n)$

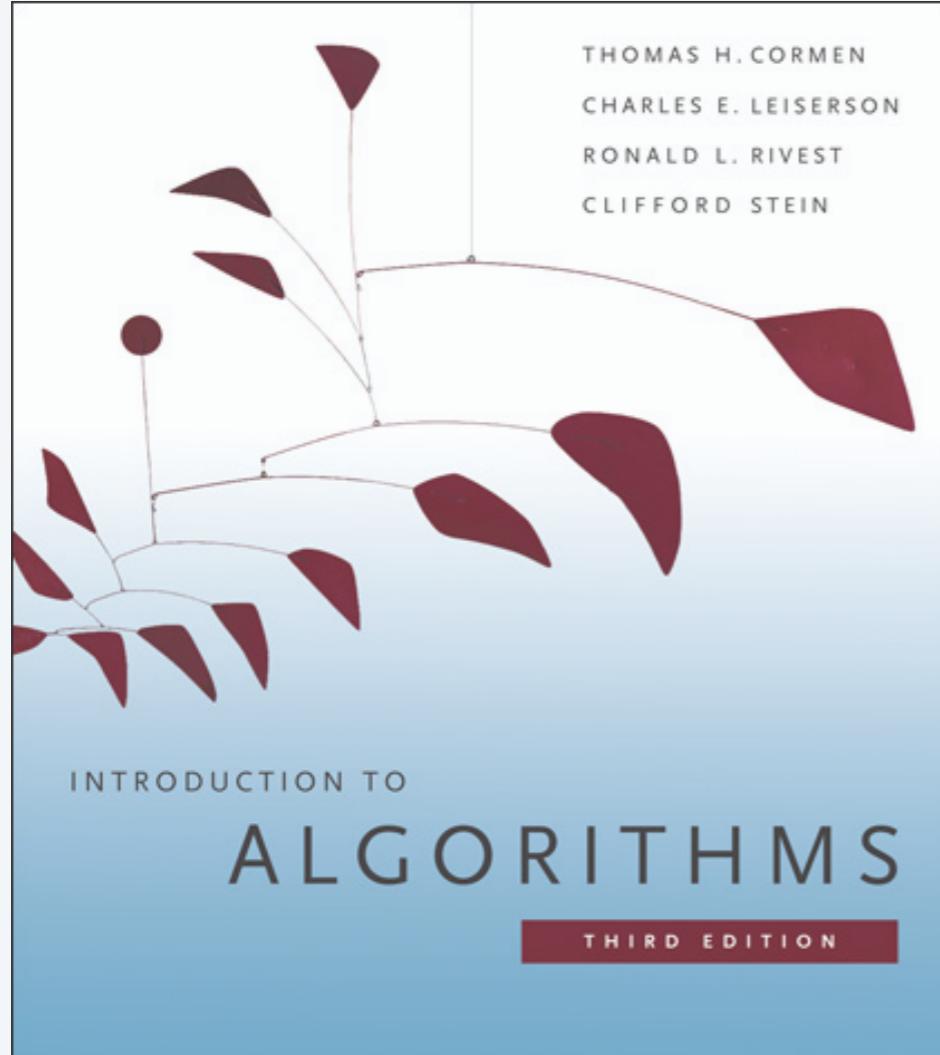
RETURN $(r_A + r_B + r_{AB}, L)$.

Counting inversions: divide-and-conquer algorithm analysis

Proposition. The sort-and-count algorithm counts the number of inversions in a permutation of size n in $O(n \log n)$ time.

Pf. The worst-case running time $T(n)$ satisfies the recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$



SECTION 7.1-7.3

5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *randomized quicksort*
- ▶ *median and selection*
- ▶ *closest pair of points*

3-WAY PARTITIONING



Goal. Given an array A and pivot element p , partition array so that:

- Smaller elements in left subarray L .
- Equal elements in middle subarray M .
- Larger elements in right subarray R .

Challenge. $O(n)$ time and $O(1)$ space.



the array A



the partitioned array A



Randomized quicksort

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recursively sort both L and R .



Randomized quicksort

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recursively sort both L and R .

RANDOMIZED-QUICKSORT(A)

IF (array A has zero or one element)

RETURN.

Pick pivot $p \in A$ uniformly at random.

$(L, M, R) \leftarrow \text{PARTITION-3-WAY}(A, p)$. $\longleftarrow \Theta(n)$

$\text{RANDOMIZED-QUICKSORT}(L)$. $\longleftarrow T(i)$

$\text{RANDOMIZED-QUICKSORT}(R)$. $\longleftarrow T(n - i - 1)$

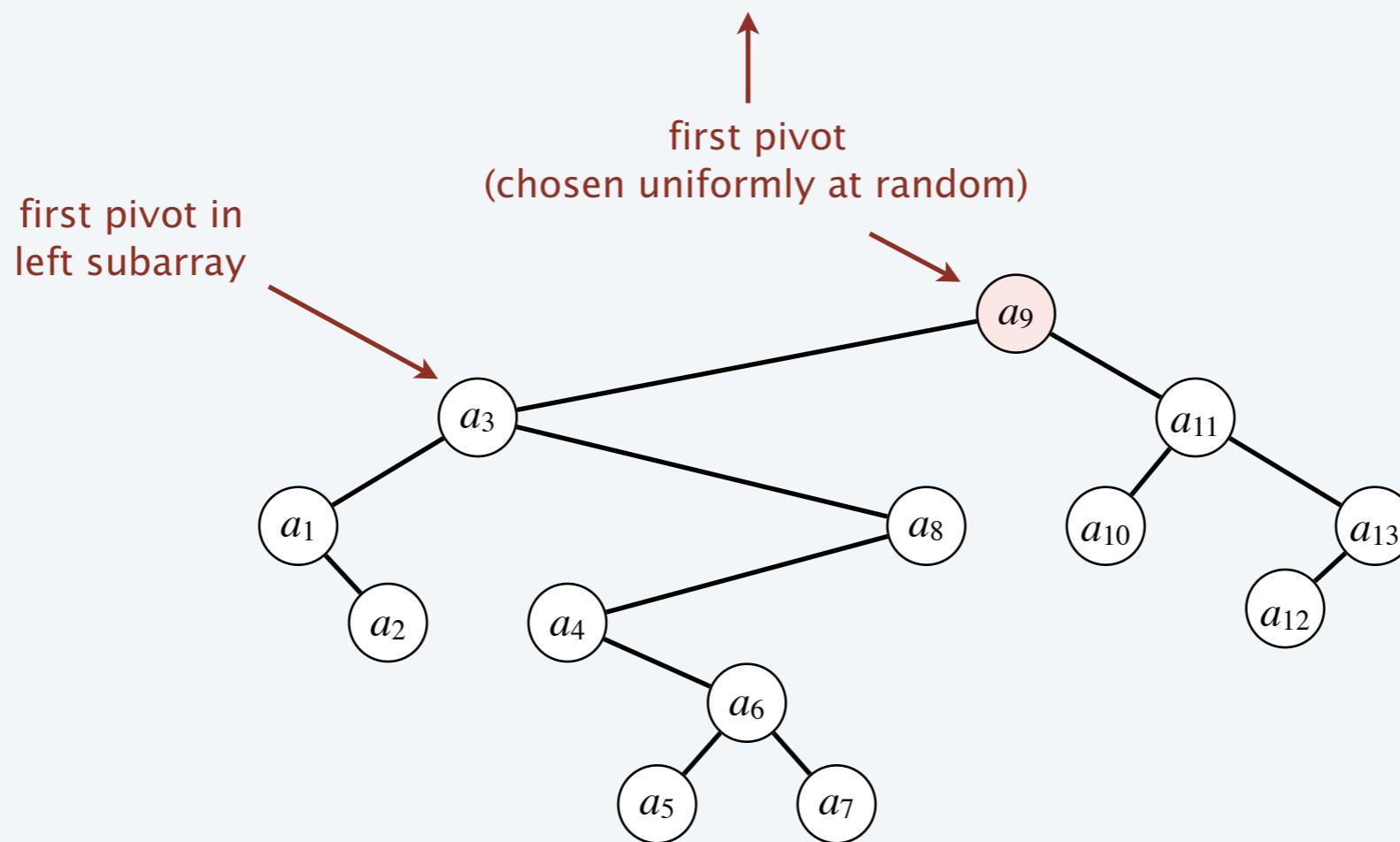
[new analysis required
 i is a random variable—depends on p]

Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

the original array of elements A

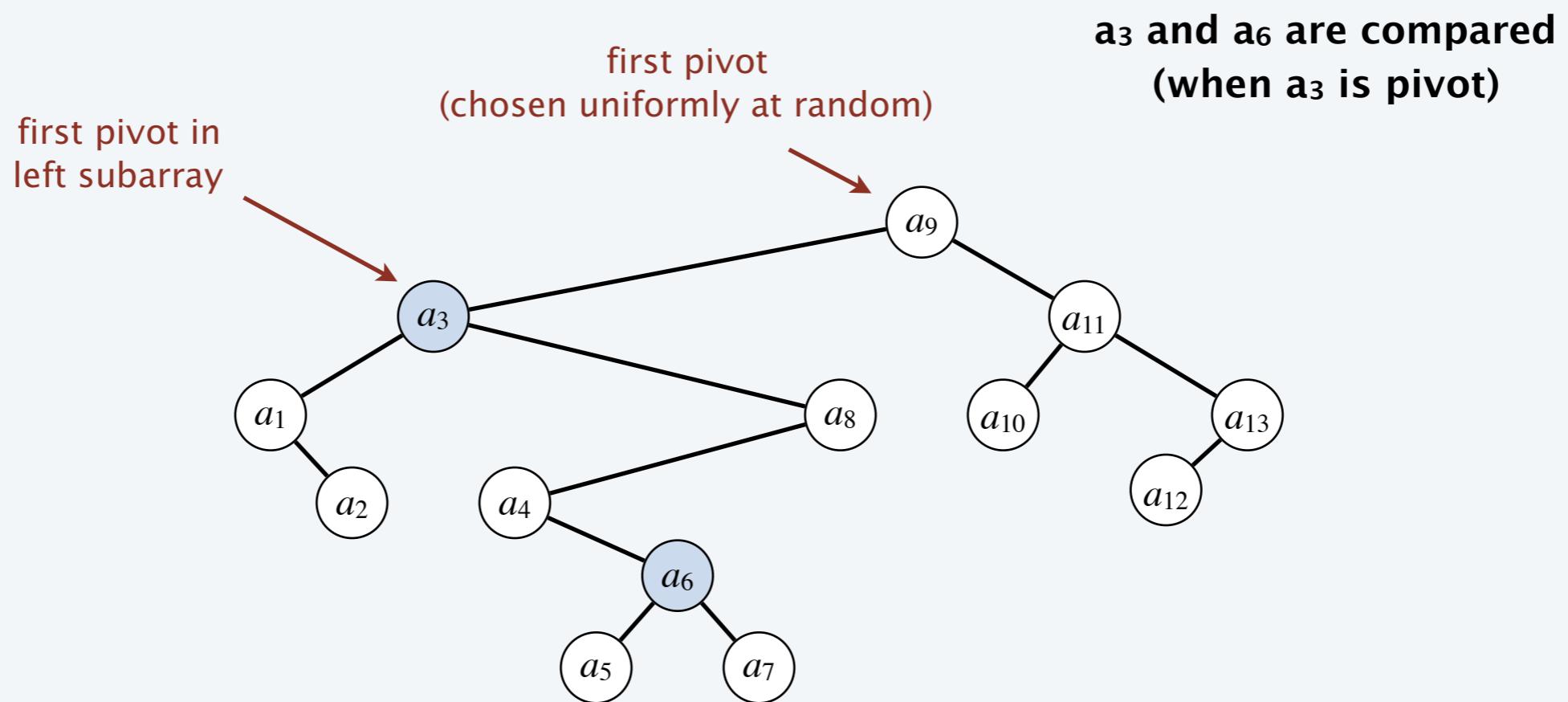


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.

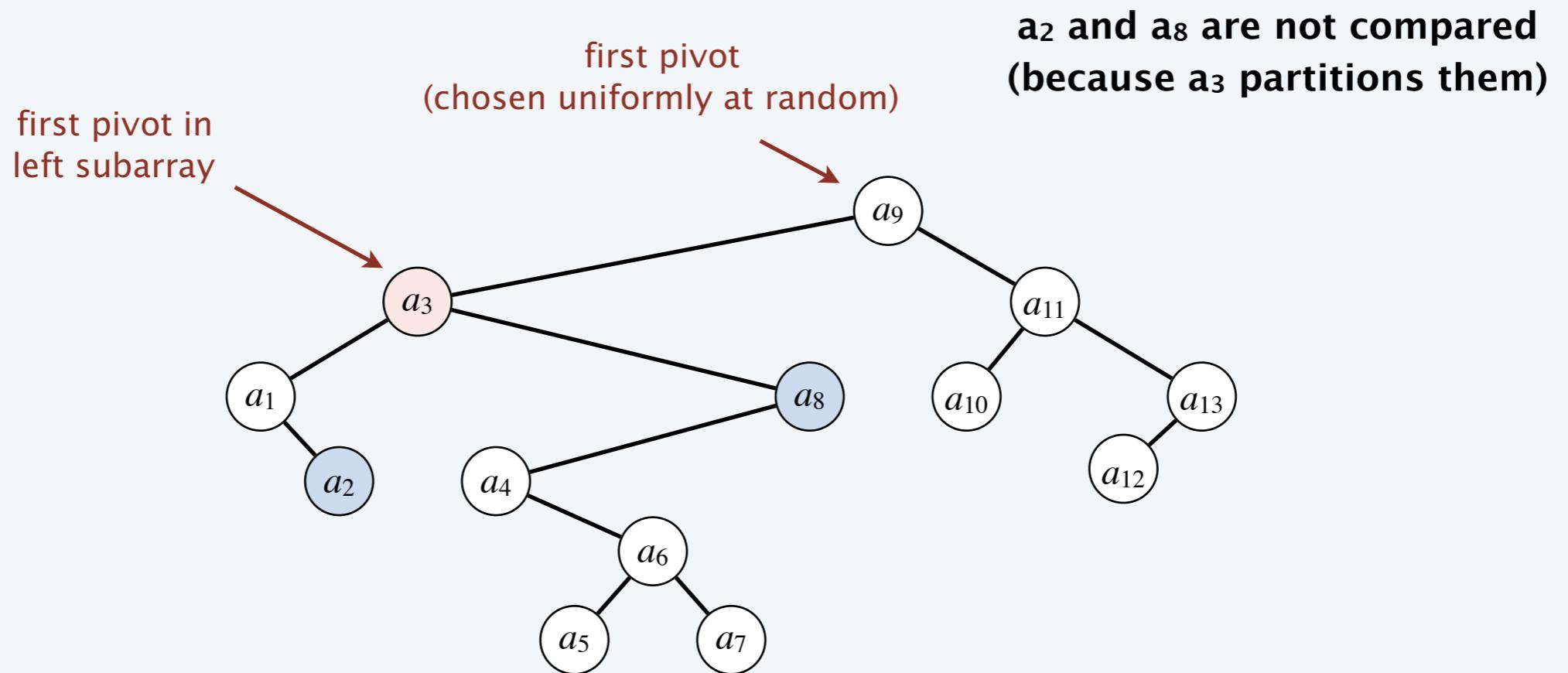


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

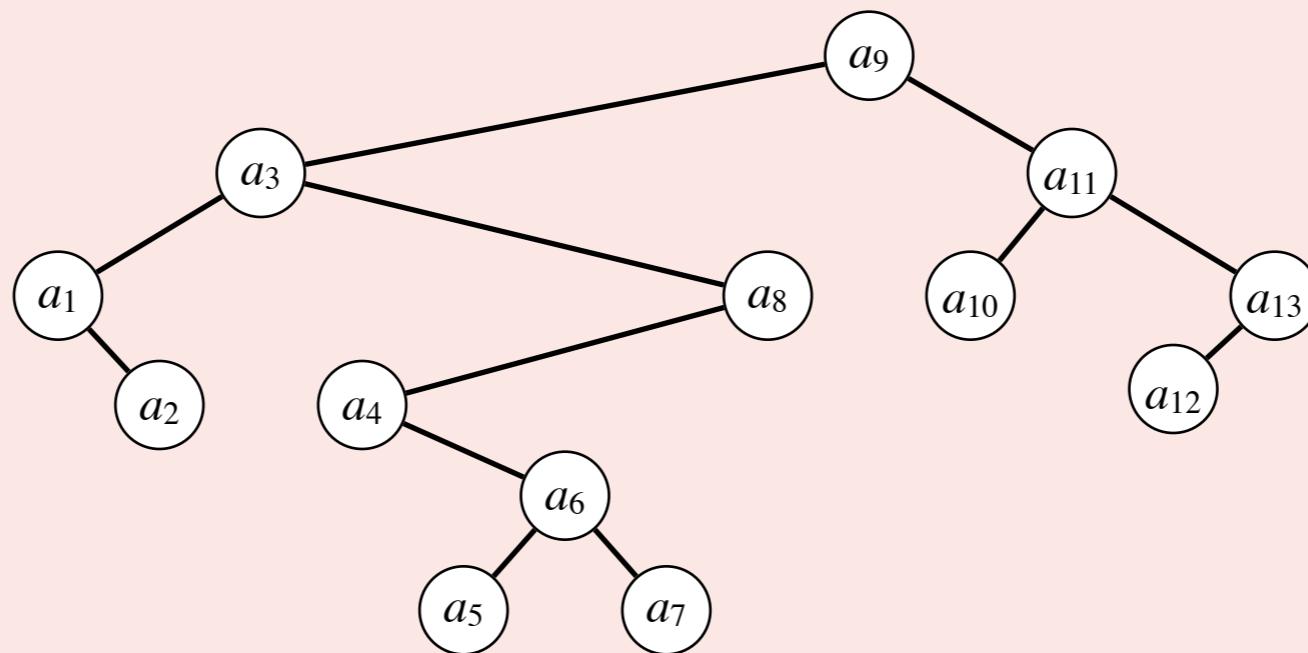
- a_i and a_j are compared once iff one is an ancestor of the other.





Given an array of $n \geq 8$ distinct elements $a_1 < a_2 < \dots < a_n$, what is the probability that a_7 and a_8 are compared during randomized quicksort?

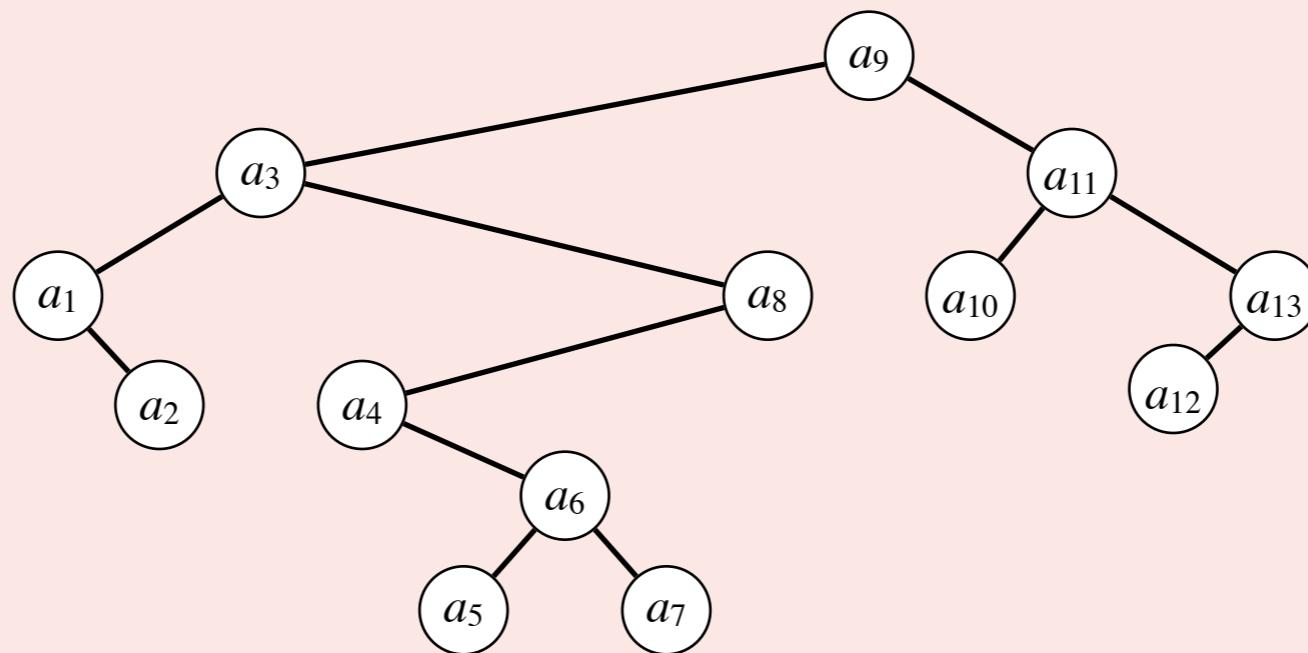
- A. 0
- B. $1 / n$
- C. $2 / n$
- D. 1





Given an array of $n \geq 2$ distinct elements $a_1 < a_2 < \dots < a_n$, what is the probability that a_1 and a_n are compared during randomized quicksort?

- A. 0
- B. $1 / n$
- C. $2 / n$
- D. 1



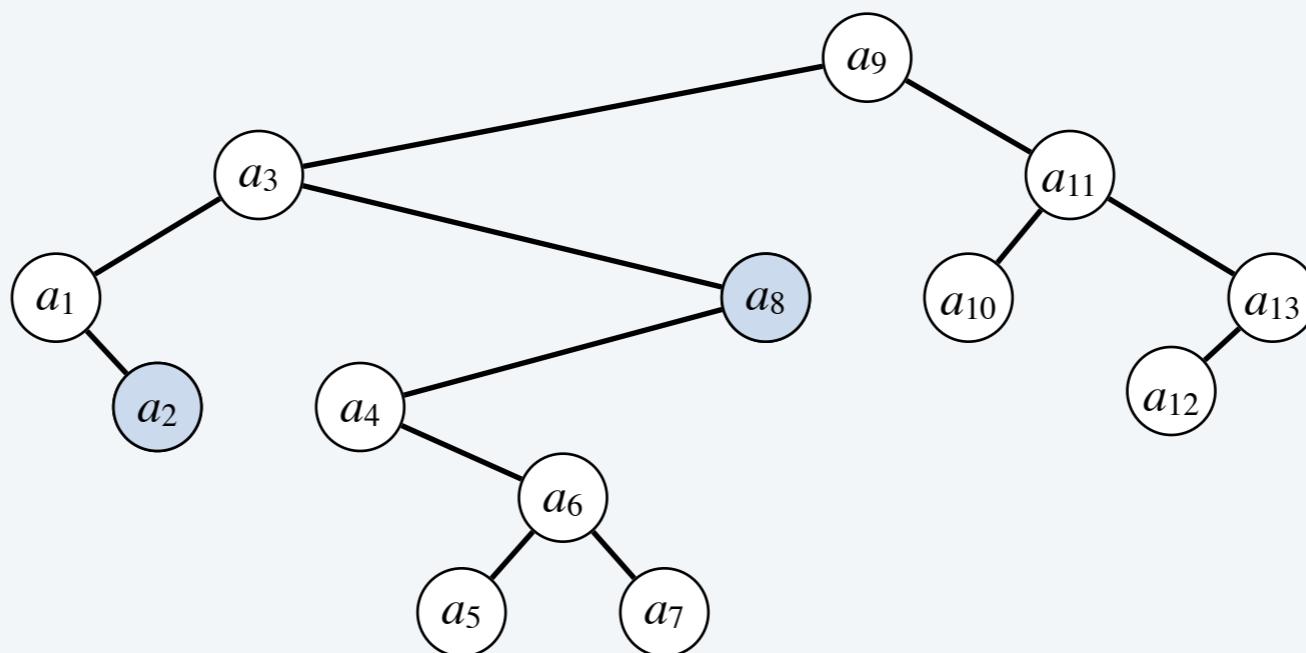
Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.
- $\Pr [a_i \text{ and } a_j \text{ are compared}] = 2 / (j - i + 1)$, where $i < j$.

$\Pr[a_2 \text{ and } a_8 \text{ compared}] = 2/7$
compared iff either a_2 or a_8 is chosen
as pivot before any of $\{a_3, a_4, a_5, a_6, a_7\}$



Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.
- $\Pr [a_i \text{ and } a_j \text{ are compared}] = 2 / (j - i + 1)$, where $i < j$.
- Expected number of compares $= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j - i + 1} = 2 \sum_{i=1}^n \sum_{j=2}^{n-i+1} \frac{1}{j}$

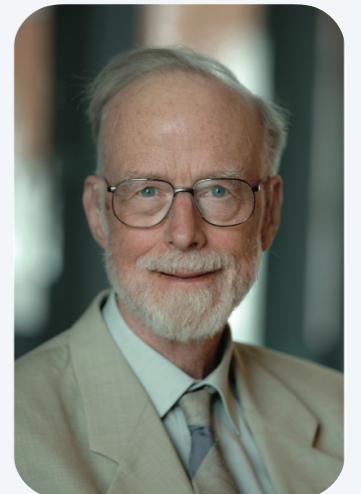
all pairs i and j
 $\leq 2n \sum_{j=1}^n \frac{1}{j}$
 $\leq 2n (\ln n + 1)$ ■

harmonic sum

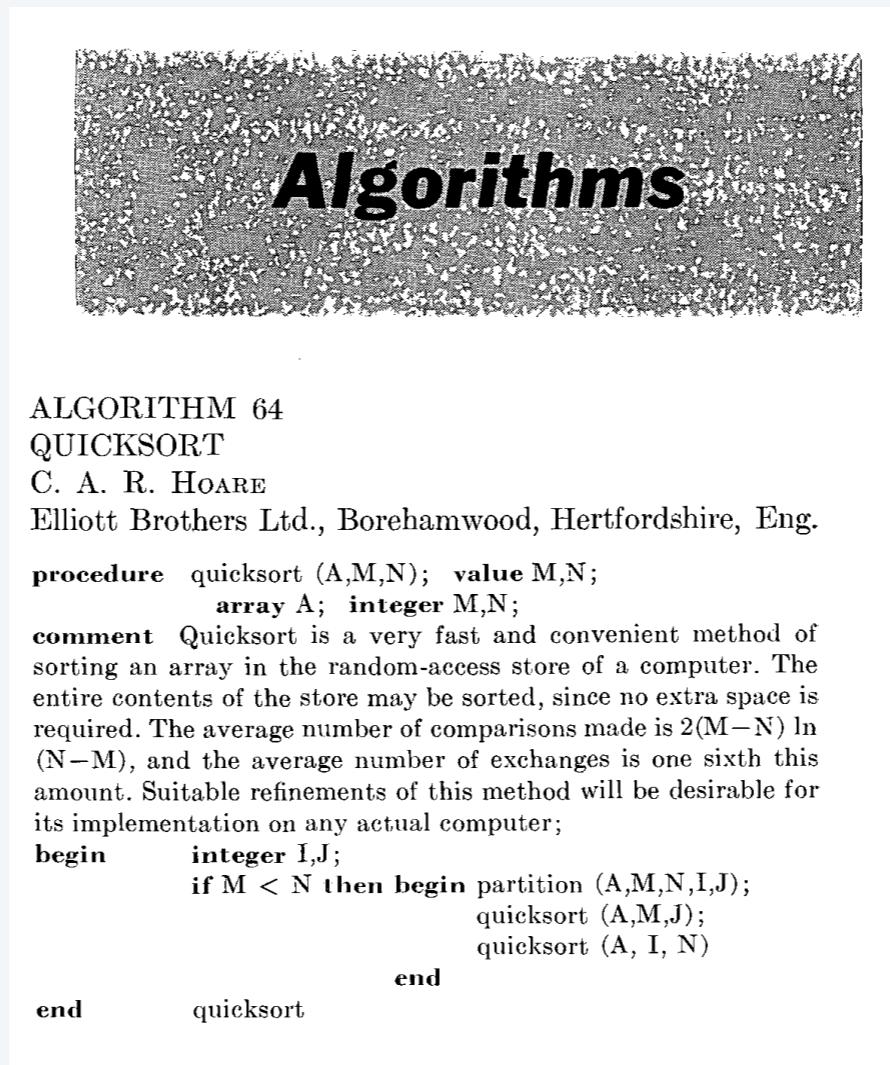
Remark. Number of compares only decreases if equal elements.

Tony Hoare

- Invented quicksort to translate Russian into English.
[but couldn't explain his algorithm or implement it!]
- Learned Algol 60 (and recursion).
- Implemented quicksort.



Tony Hoare
1980 Turing Award



ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure quicksort (A,M,N); value M,N;
    array A; integer M,N;
comment Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is  $2(M-N) \ln$ 
 $(N-M)$ , and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin      integer I,J;
            if M < N then begin partition (A,M,N,I,J);
                           quicksort (A,M,J);
                           quicksort (A, I, N)
                         end
end      quicksort
```

Communications of the ACM (July 1961)

NUTS AND BOLTS



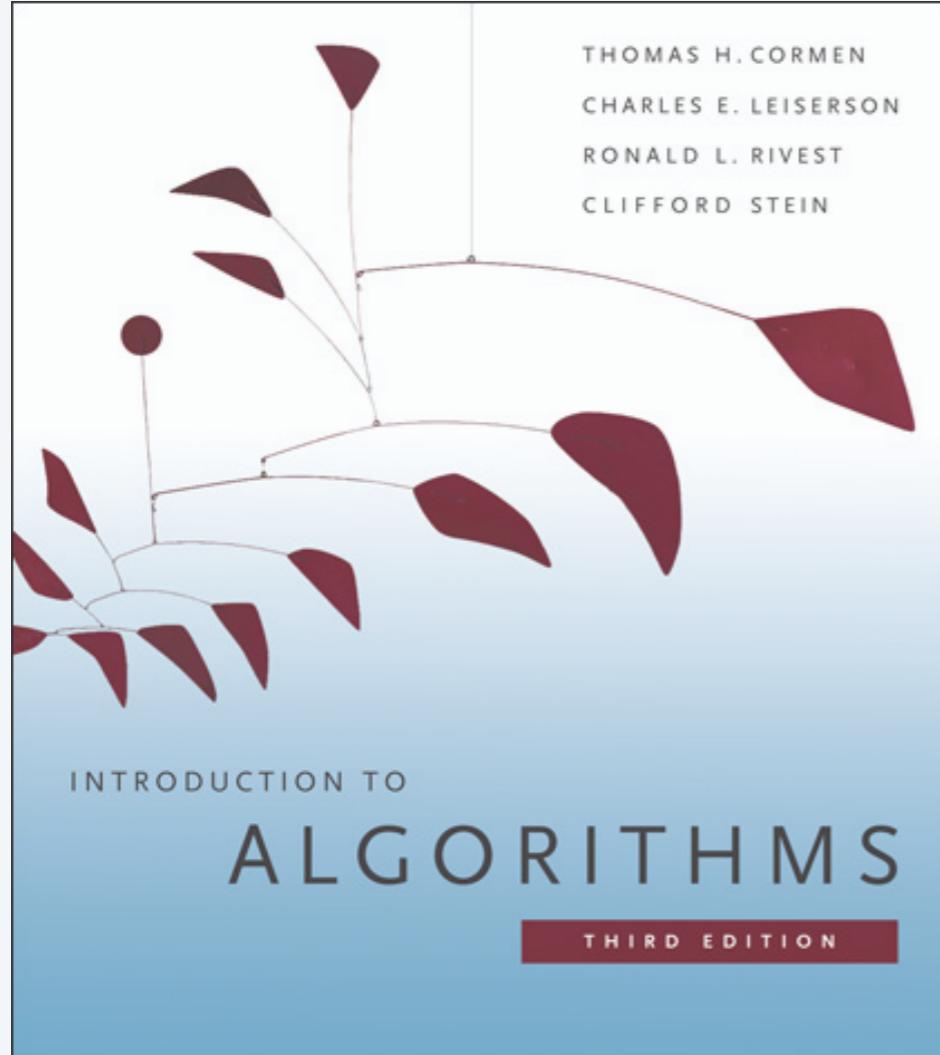
Problem. A disorganized carpenter has a mixed pile of n nuts and n bolts.

- The goal is to find the corresponding pairs of nuts and bolts.
- Each nut fits exactly one bolt and each bolt fits exactly one nut.
- By fitting a nut and a bolt together, the carpenter can see which one is bigger (but cannot directly compare either two nuts or two bolts).



Brute-force solution. Compare each bolt to each nut— $\Theta(n^2)$ compares.

Challenge. Design an algorithm that makes $O(n \log n)$ compares.



SECTION 9.3

5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *randomized quicksort*
- ▶ *median and selection*
- ▶ *closest pair of points*

Median and selection problems

Selection. Given n elements from a totally ordered universe, find k^{th} smallest.

- Minimum: $k = 1$; maximum: $k = n$.
- Median: $k = \lfloor (n + 1) / 2 \rfloor$.
- $O(n)$ compares for min or max.
- $O(n \log n)$ compares by sorting.
- $O(n \log k)$ compares with a binary heap. ← max heap with k smallest

Applications. Order statistics; find the “top k ”; bottleneck paths, ...

Q. Can we do it with $O(n)$ compares?

A. Yes! Selection is easier than sorting.

Randomized quickselect

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in one subarray—the one containing the k^{th} smallest element.



QUICK-SELECT(A, k)

Pick pivot $p \in A$ uniformly at random.

$(L, M, R) \leftarrow \text{PARTITION-3-WAY}(A, p)$. $\longleftarrow \Theta(n)$

IF $(k \leq |L|)$ RETURN QUICK-SELECT(L, k). $\longleftarrow T(i)$

ELSE IF $(k > |L| + |M|)$ RETURN QUICK-SELECT($R, k - |L| - |M|$) $\longleftarrow T(n - i - 1)$

ELSE IF $(k = |L|)$ RETURN p .

Randomized quickselect analysis

Intuition. Split candy bar uniformly \Rightarrow expected size of larger piece is $\frac{3}{4}$.

$$T(n) \leq T(3n/4) + n \Rightarrow T(n) \leq 4n$$

not rigorous: can't assume
 $E[T(i)] \leq T(E[i])$



Def. $T(n, k)$ = expected # compares to select k^{th} smallest in array of length $\leq n$.

Def. $T(n) = \max_k T(n, k)$.

Proposition. $T(n) \leq 4n$.

Pf. [by strong induction on n]

- Assume true for $1, 2, \dots, n-1$.
- $T(n)$ satisfies the following recurrence:

can assume we always recur of
larger of two subarrays since $T(n)$
is monotone non-decreasing

$$T(n) \leq n + 1/n [2T(n/2) + \dots + 2T(n-3) + 2T(n-2) + 2T(n-1)]$$

$$\leq n + 1/n [8(n/2) + \dots + 8(n-3) + 8(n-2) + 8(n-1)]$$

$$\leq n + 1/n (3n^2)$$

$$= 4n. \blacksquare$$

inductive hypothesis

tiny cheat: sum should start at $T(\lfloor n/2 \rfloor)$

Selection in worst-case linear time

Goal. Find pivot element p that divides list of n elements into two pieces so that each piece is **guaranteed** to have $\leq 7/10 n$ elements.

Q. How to find approximate median in linear time?

A. Recursively compute median of sample of $\leq 2/10 n$ elements.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(7/10 n) + T(2/10 n) + \Theta(n) & \text{otherwise} \end{cases}$$

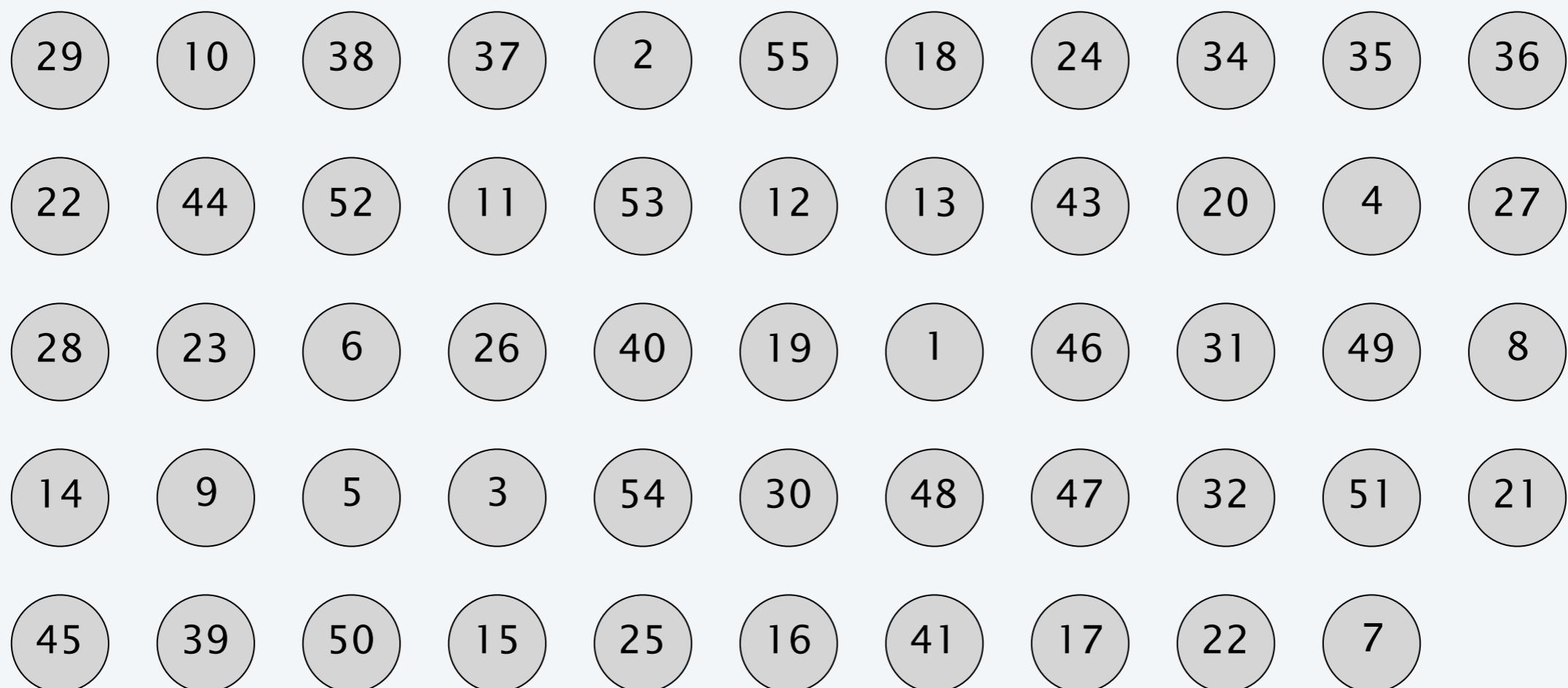
two subproblems
of different sizes!

$$\Rightarrow T(n) = \Theta(n)$$

we'll need to show this

Choosing the pivot element

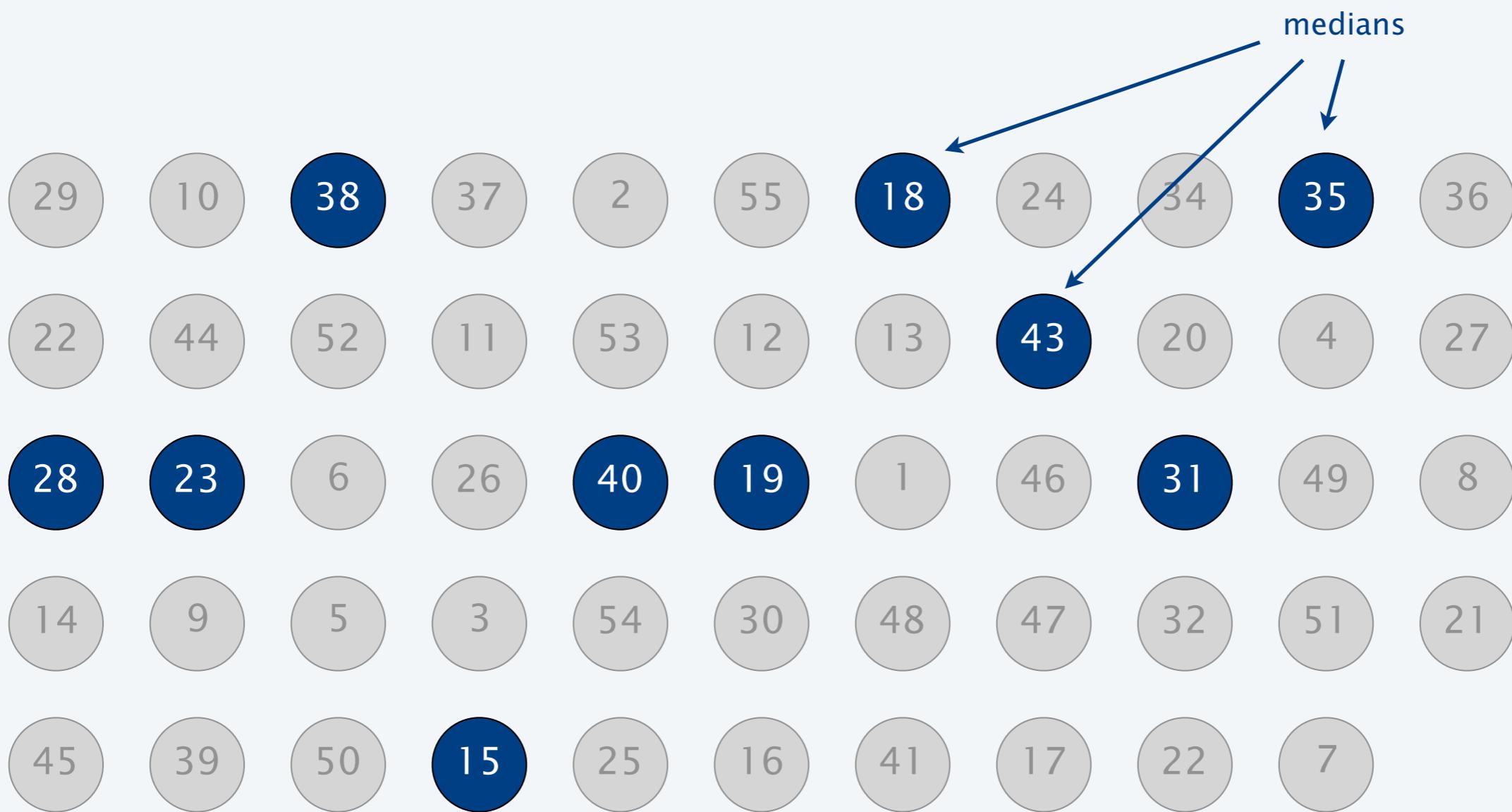
- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).



$n = 54$

Choosing the pivot element

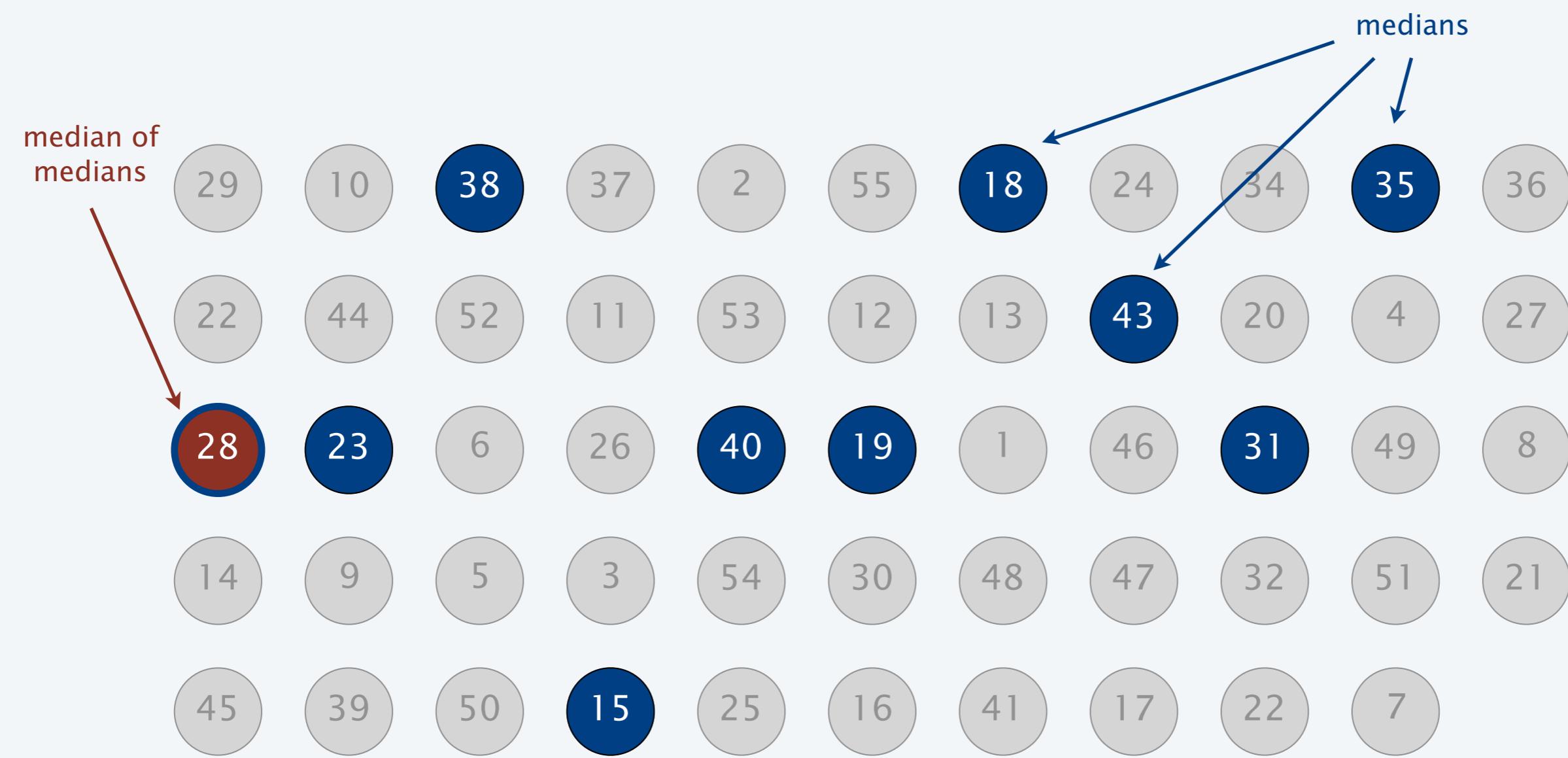
- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).



$n = 54$

Choosing the pivot element

- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).
- Find median of $\lfloor n / 5 \rfloor$ medians recursively.
- Use median-of-medians as pivot element.



$n = 54$

Median-of-medians selection algorithm

MOM-SELECT(A, k)

$n \leftarrow |A|.$

IF ($n < 50$)

RETURN k^{th} smallest of element of A via mergesort.

Group A into $\lfloor n / 5 \rfloor$ groups of 5 elements each (ignore leftovers).

$B \leftarrow$ median of each group of 5.

$p \leftarrow$ **MOM-SELECT**($B, \lfloor n / 10 \rfloor$) \longleftarrow median of medians

$(L, M, R) \leftarrow$ PARTITION-3-WAY(A, p).

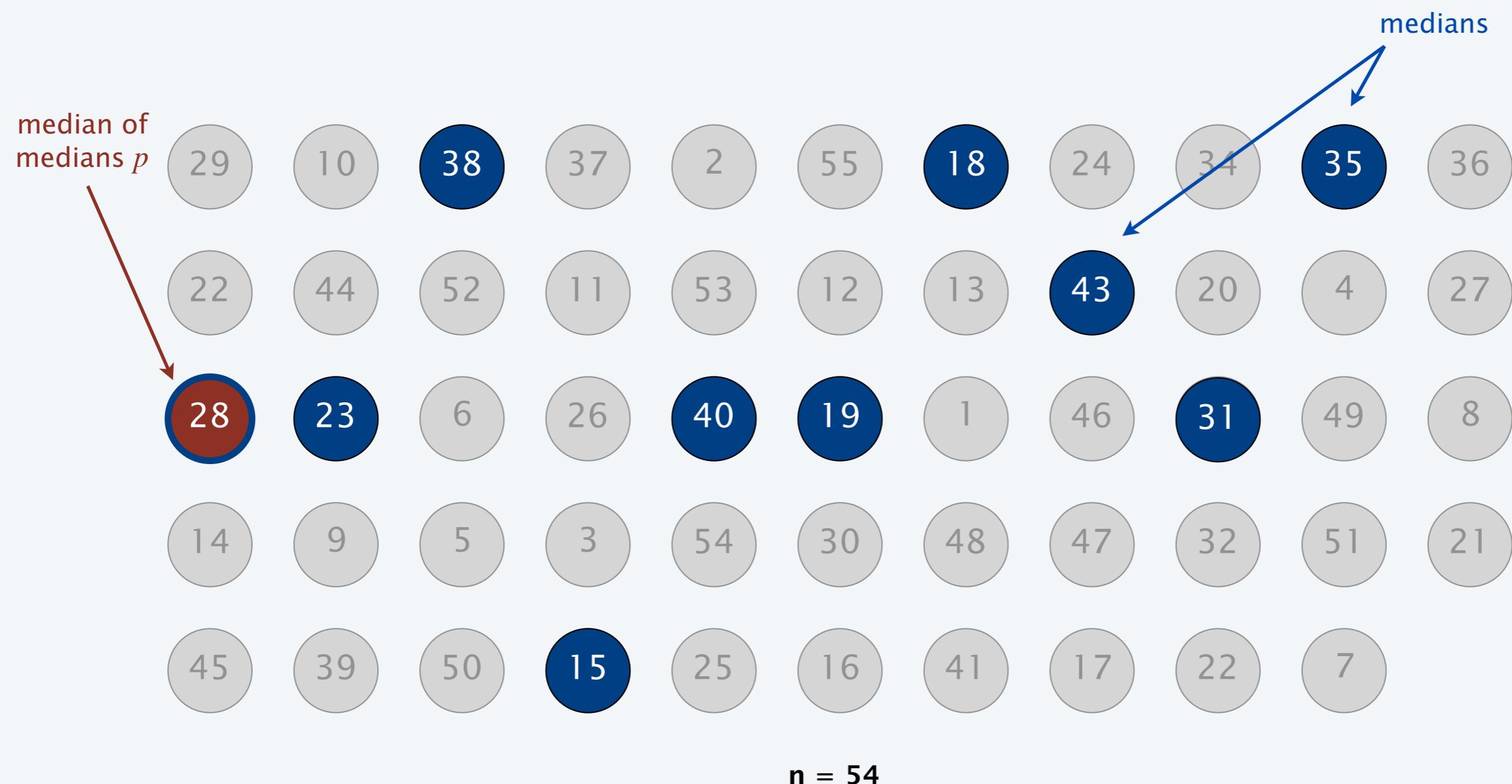
IF $(k \leq |L|)$ **RETURN** **MOM-SELECT**(L, k).

ELSE IF $(k > |L| + |M|)$ **RETURN** **MOM-SELECT**($R, k - |L| - |M|$)

ELSE **RETURN** p .

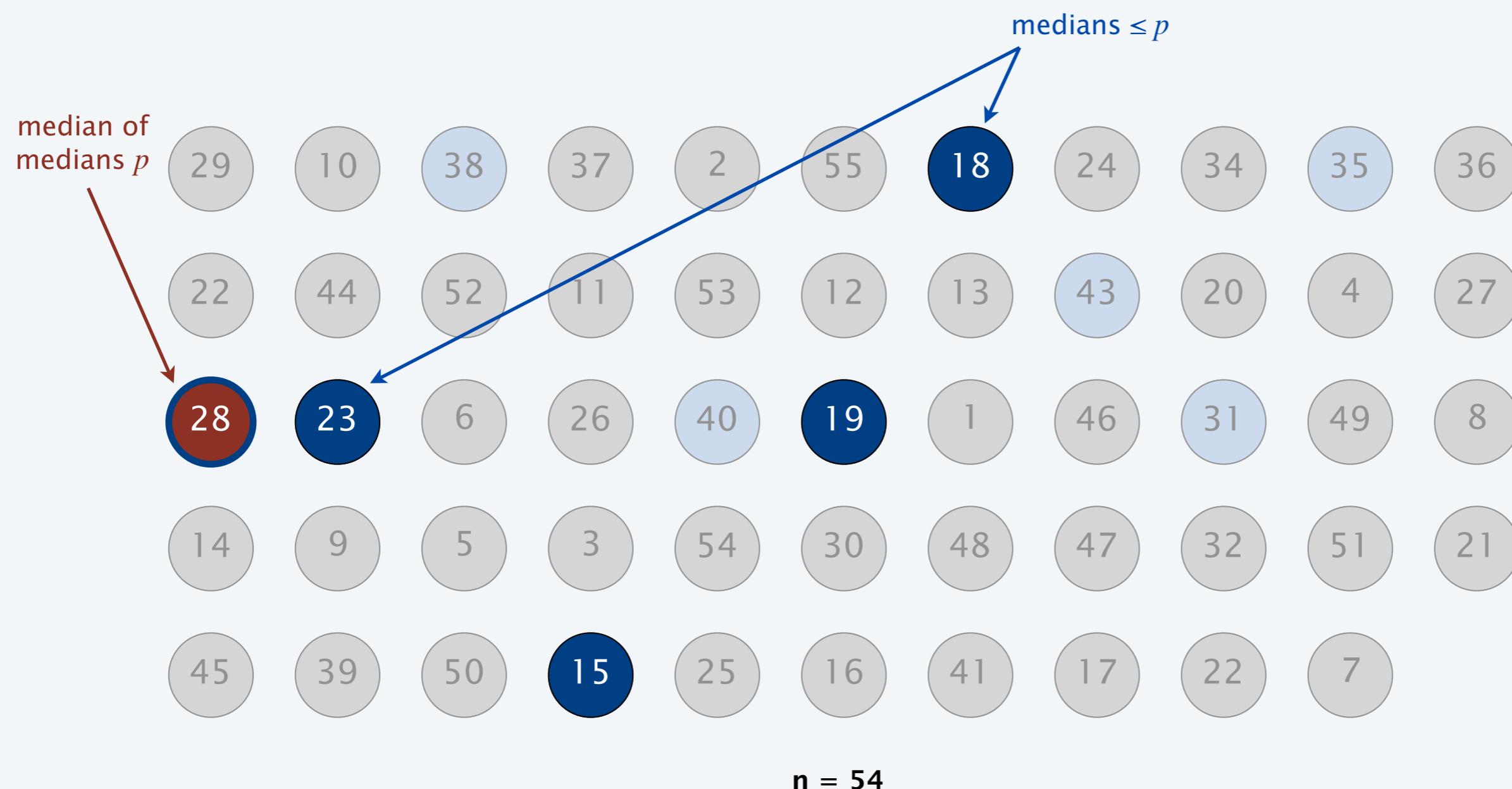
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\leq p$.



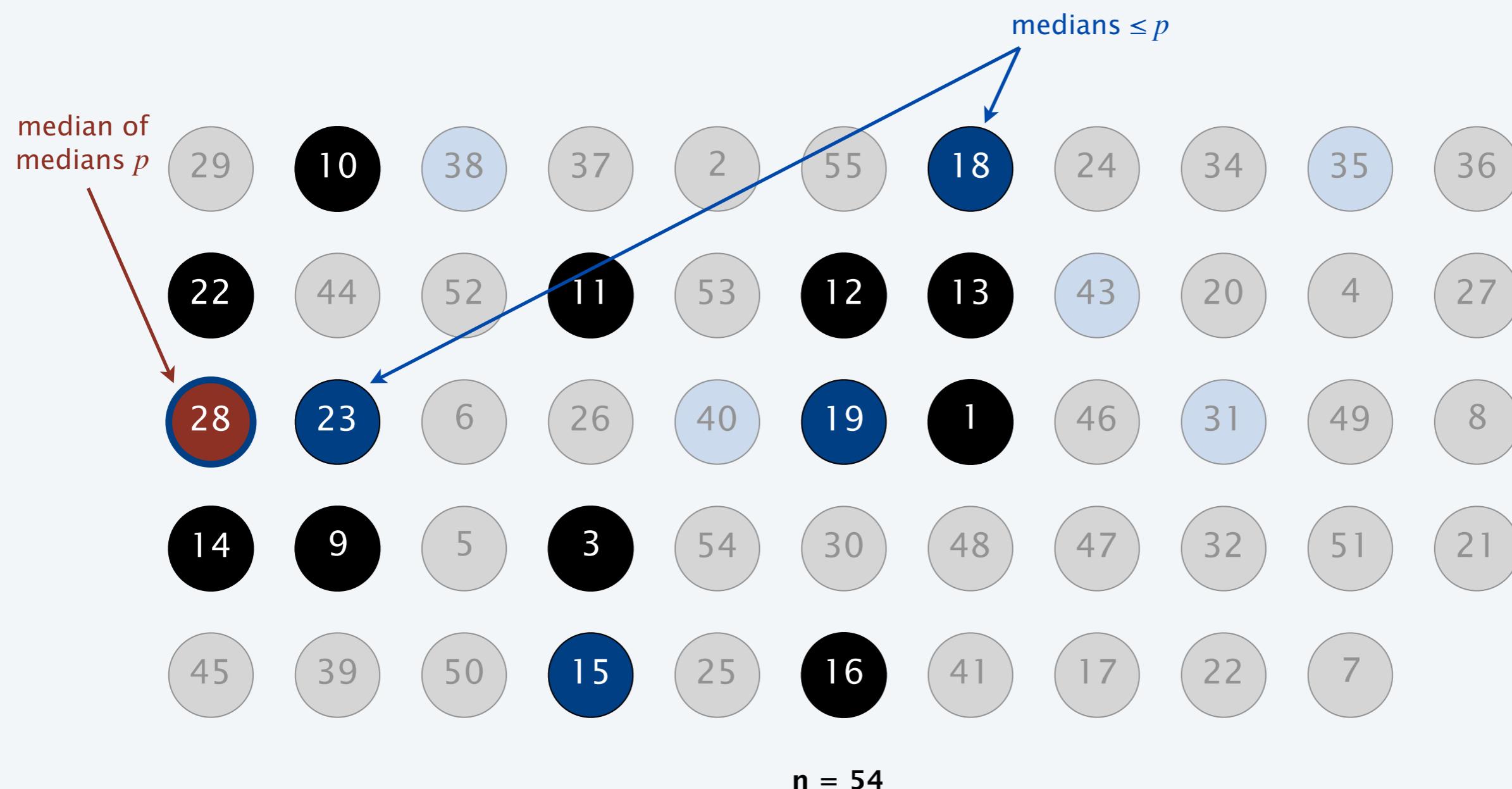
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\leq p$.
- At least $\lfloor [n/5]/2 \rfloor = \lfloor n/10 \rfloor$ medians $\leq p$.



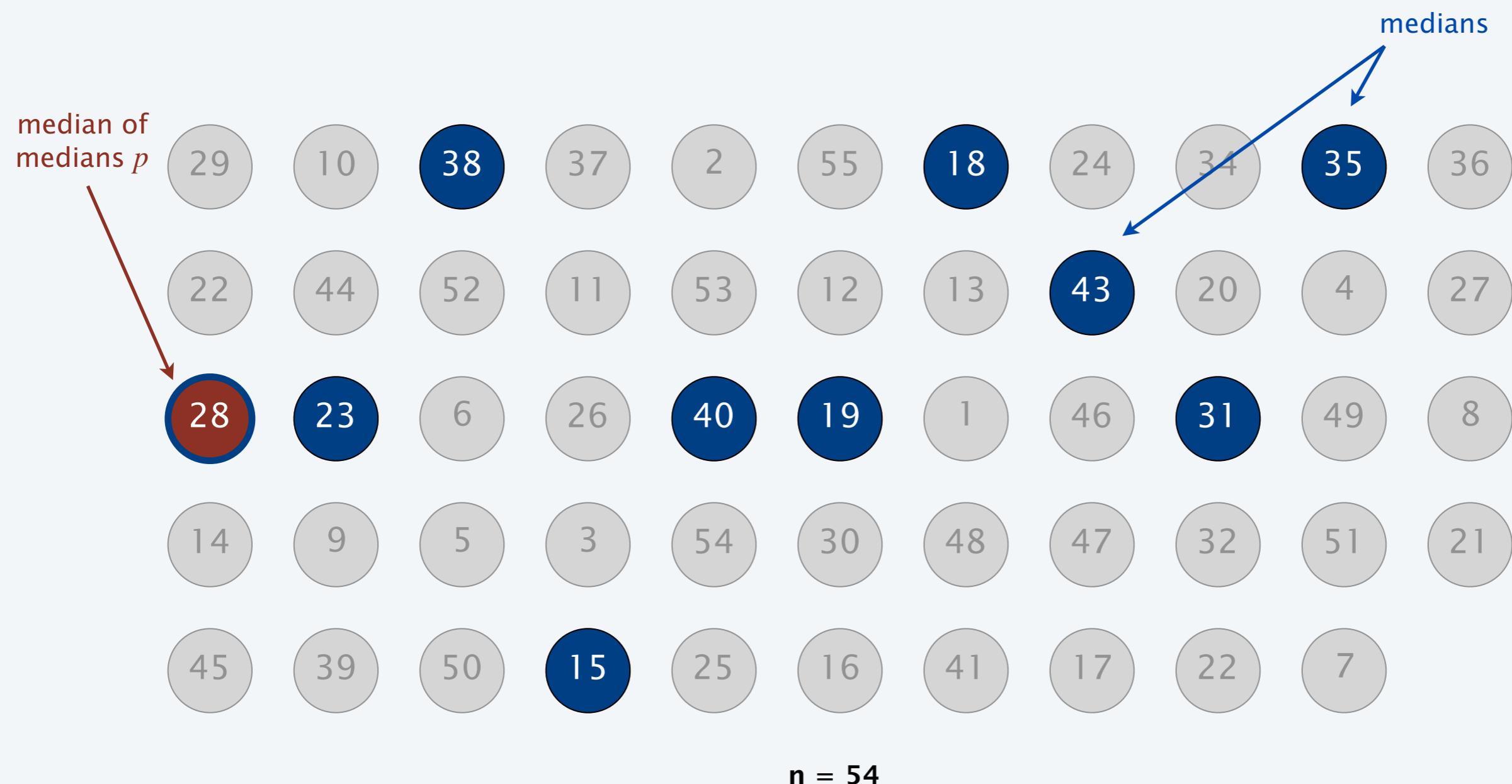
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\leq p$.
- At least $\lfloor [n/5]/2 \rfloor = \lfloor n/10 \rfloor$ medians $\leq p$.
- At least $3\lfloor n/10 \rfloor$ elements $\leq p$.



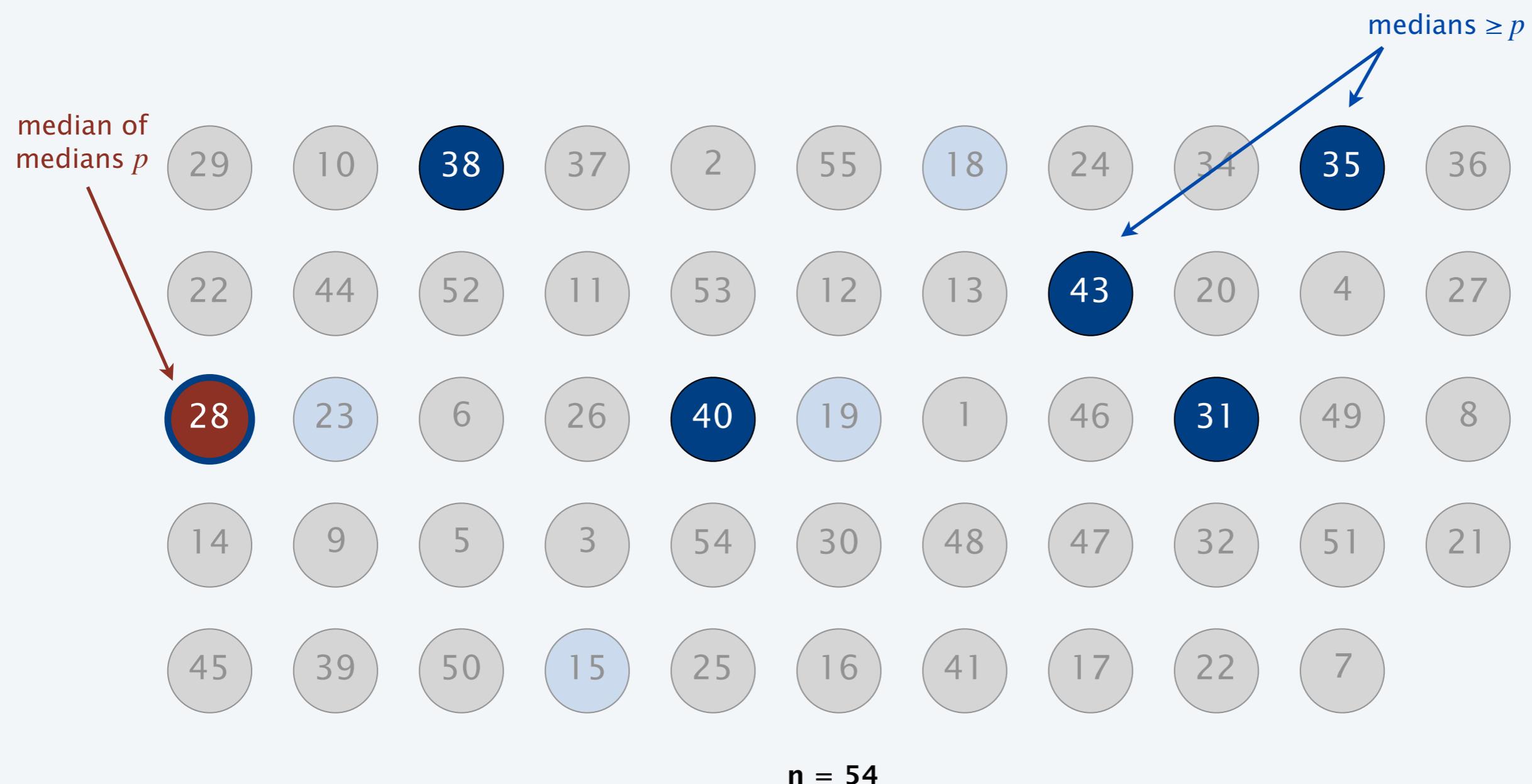
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\geq p$.



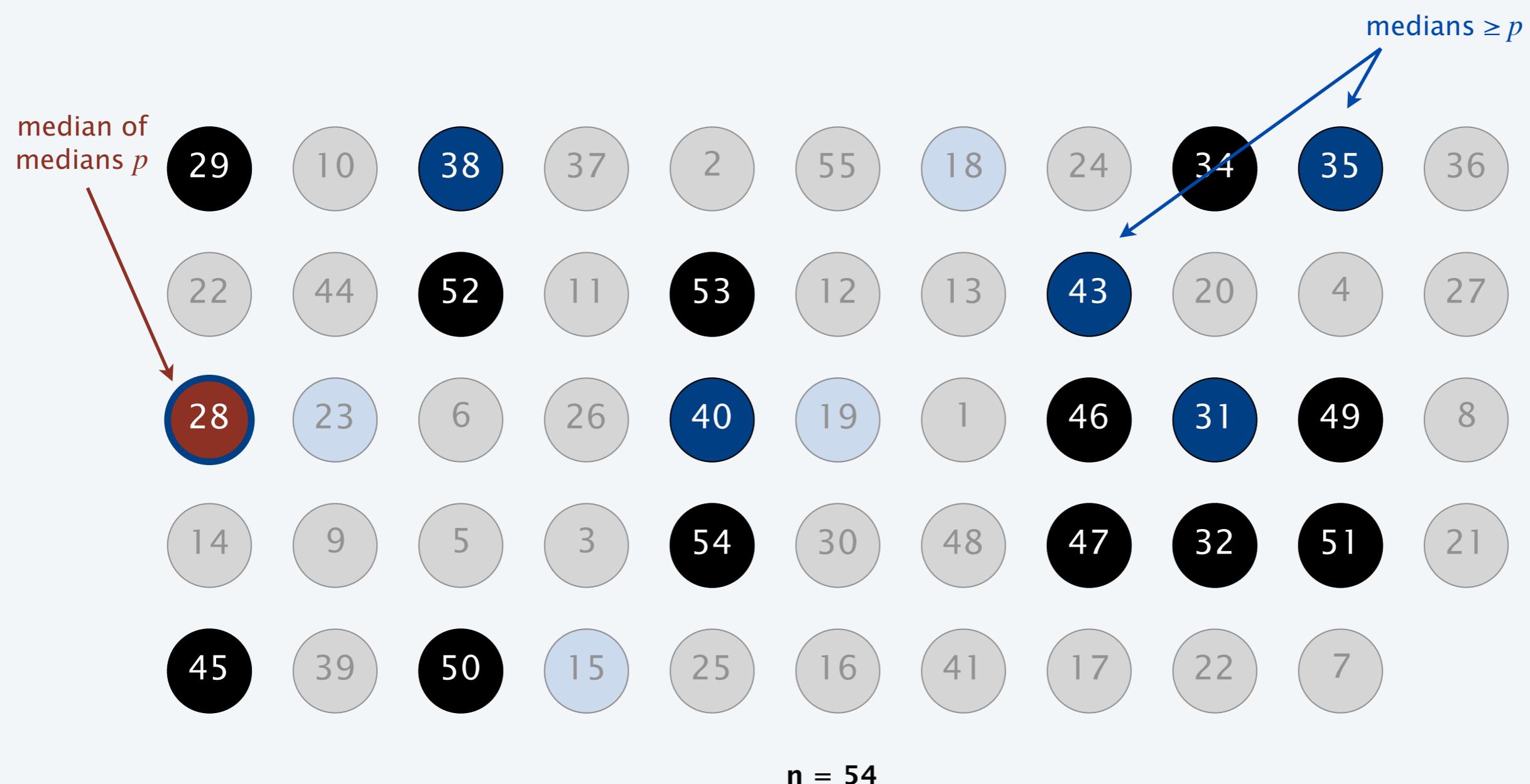
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\geq p$.
- At least $\lfloor [n/5]/2 \rfloor = \lfloor n/10 \rfloor$ medians $\geq p$.



Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\geq p$.
- At least $\lfloor [n/5]/2 \rfloor = \lfloor n/10 \rfloor$ medians $\geq p$.
- At least $3\lfloor n/10 \rfloor$ elements $\geq p$.



Median-of-medians selection algorithm recurrence

Median-of-medians selection algorithm recurrence.

- Select called recursively with $\lfloor n / 5 \rfloor$ elements to compute MOM p .
- At least $3 \lfloor n / 10 \rfloor$ elements $\leq p$.
- At least $3 \lfloor n / 10 \rfloor$ elements $\geq p$.
- Select called recursively with at most $n - 3 \lfloor n / 10 \rfloor$ elements.

Def. $C(n) = \max \# \text{ compares on any array of } n \text{ elements.}$

$$C(n) \leq C(\lfloor n/5 \rfloor) + C(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n$$

median of
medians recursive
select computing median of 5
(≤ 6 compares per group)
partitioning
($\leq n$ compares)

Intuition.

- $C(n)$ is going to be at least linear in $n \Rightarrow C(n)$ is super-additive.
- Ignoring floors, this implies that
$$\begin{aligned} C(n) &\leq C(n/5 + n - 3n/10) + 11/5 n \\ &= C(9n/10) + 11/5 n \\ &\Rightarrow C(n) \leq 22n. \end{aligned}$$

Median-of-medians selection algorithm recurrence

Median-of-medians selection algorithm recurrence.

- Select called recursively with $\lfloor n / 5 \rfloor$ elements to compute MOM p .
- At least $3 \lfloor n / 10 \rfloor$ elements $\leq p$.
- At least $3 \lfloor n / 10 \rfloor$ elements $\geq p$.
- Select called recursively with at most $n - 3 \lfloor n / 10 \rfloor$ elements.

Def. $C(n) = \max \# \text{ compares on any array of } n \text{ elements.}$

$$C(n) \leq C(\lfloor n/5 \rfloor) + C(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n$$

median of
medians recursive
select computing median of 5
(≤ 6 compares per group)
partitioning
($\leq n$ compares)

Now, let's solve given recurrence.

- Assume n is both a power of 5 and a power of 10 ?
- Prove that $C(n)$ is monotone non-decreasing.



Consider the following recurrence

$$C(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ C(\lfloor n/5 \rfloor) + C(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n & \text{if } n > 1 \end{cases}$$

Is $C(n)$ monotone non-decreasing?

- A. Yes, obviously.
- B. Yes, but proof is tedious.
- C. Yes, but proof is hard.
- D. No.

Median-of-medians selection algorithm recurrence

Analysis of selection algorithm recurrence.

- $T(n) = \max$ # compares on any array of $\leq n$ elements.
- $T(n)$ is monotone non-decreasing, but $C(n)$ is not!

$$T(n) \leq \begin{cases} 6n & \text{if } n < 50 \\ \max\{ T(n-1), T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n \} & \text{if } n \geq 50 \end{cases}$$

Claim. $T(n) \leq 44n$.

Pf. [by strong induction]

- Base case: $T(n) \leq 6n$ for $n < 50$ (mergesort).
- Inductive hypothesis: assume true for $1, 2, \dots, n-1$.
- Induction step: for $n \geq 50$, we have either $T(n) \leq T(n-1) \leq 44n$ or

$$T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5n$$

inductive hypothesis $\longrightarrow \leq 44(\lfloor n/5 \rfloor) + 44(n - 3\lfloor n/10 \rfloor) + 11/5n$

$$\begin{aligned} &\leq 44(n/5) + 44n - 44(n/4) + 11/5n \quad \leftarrow \text{for } n \geq 50, 3\lfloor n/10 \rfloor \geq n/4 \\ &= 44n. \quad \blacksquare \end{aligned}$$

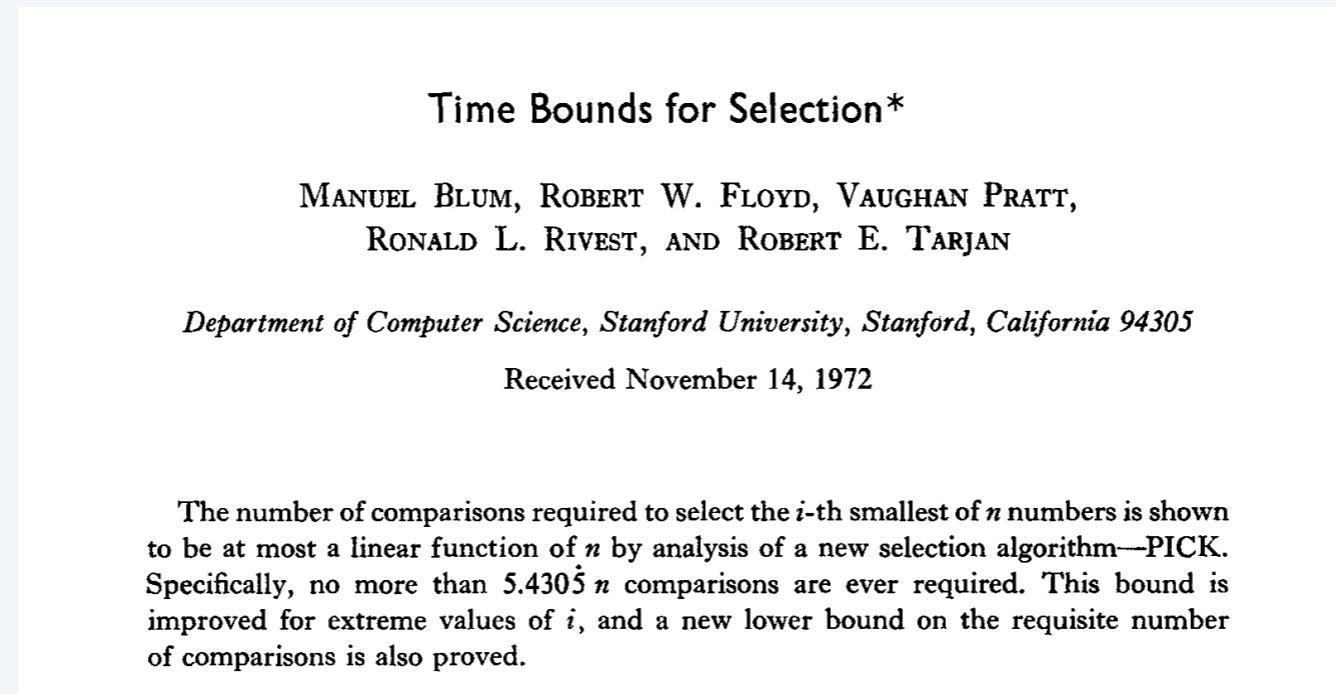


Suppose that we divide n elements into $\lfloor n / r \rfloor$ groups of r elements each, and use the median-of-medians of these $\lfloor n / r \rfloor$ groups as the pivot.
For which r is the worst-case running time of select $O(n)$?

- A. $r = 3$
- B. $r = 7$
- C. Both A and B.
- D. Neither A nor B.

Linear-time selection retrospective

Proposition. [Blum–Floyd–Pratt–Rivest–Tarjan 1973] There exists a compare-based selection algorithm whose worst-case running time is $O(n)$.

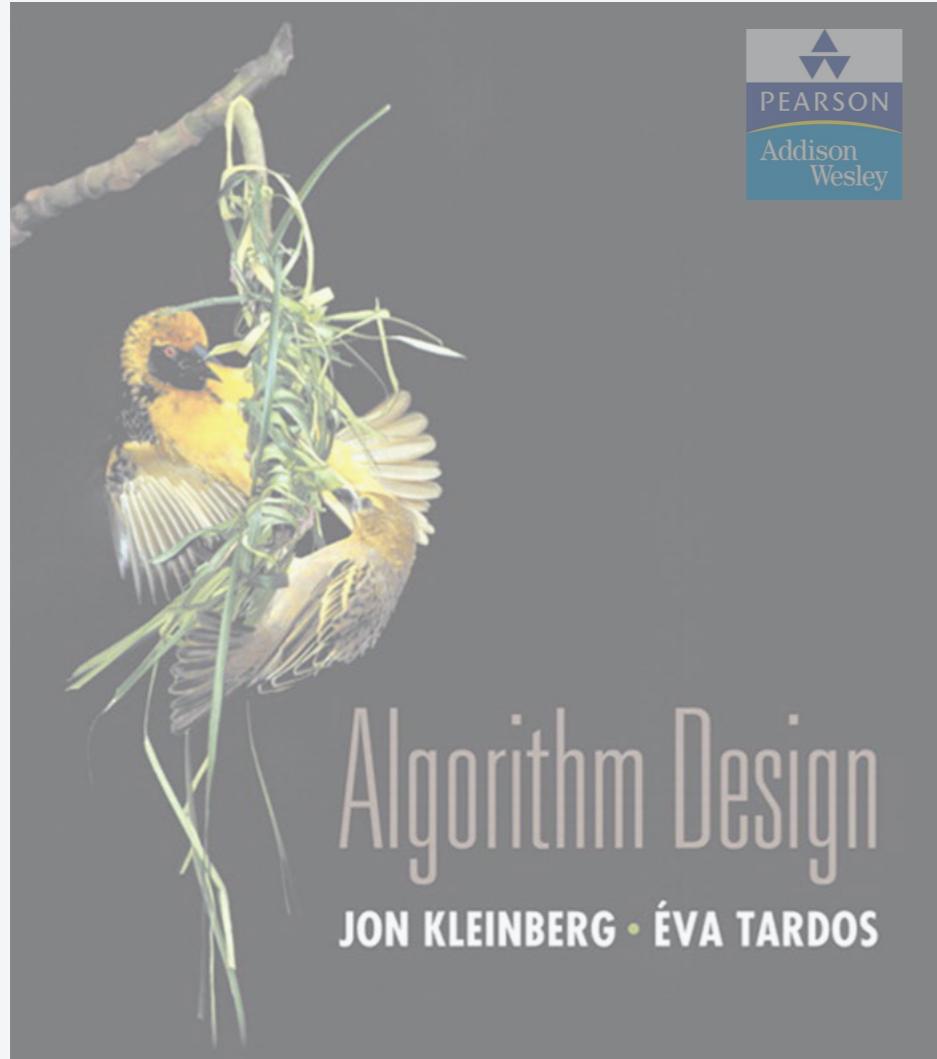


Theory.

- Optimized version of BFPRT: $\leq 5.4305 n$ compares.
- Upper bound: [Dor–Zwick 1995] $\leq 2.95 n$ compares.
- Lower bound: [Dor–Zwick 1999] $\geq (2 + 2^{-80}) n$ compares.

Practice. Constants too large to be useful.





SECTION 5.4

5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *randomized quicksort*
- ▶ *median and selection*
- ▶ *closest pair of points*

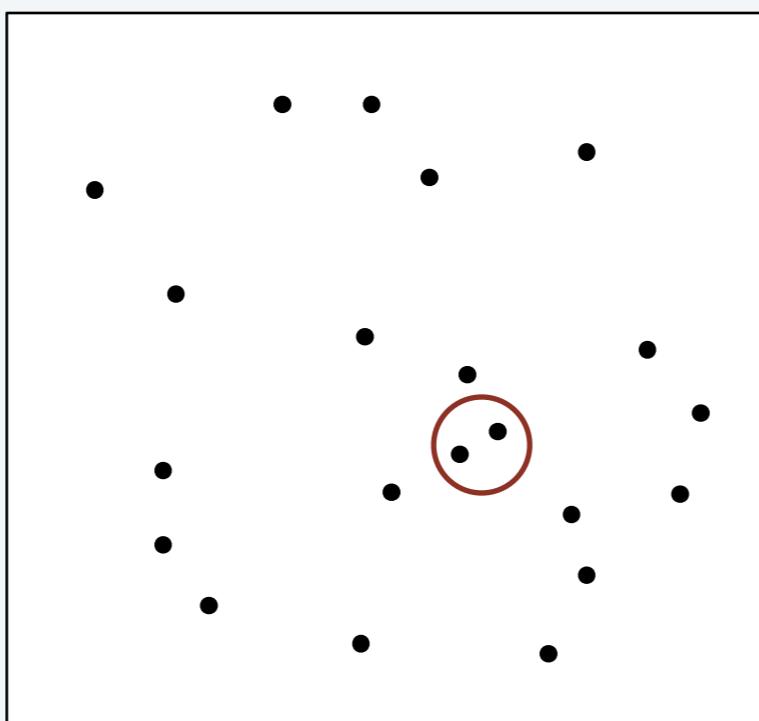
Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

fast closest pair inspired fast algorithms for these problems



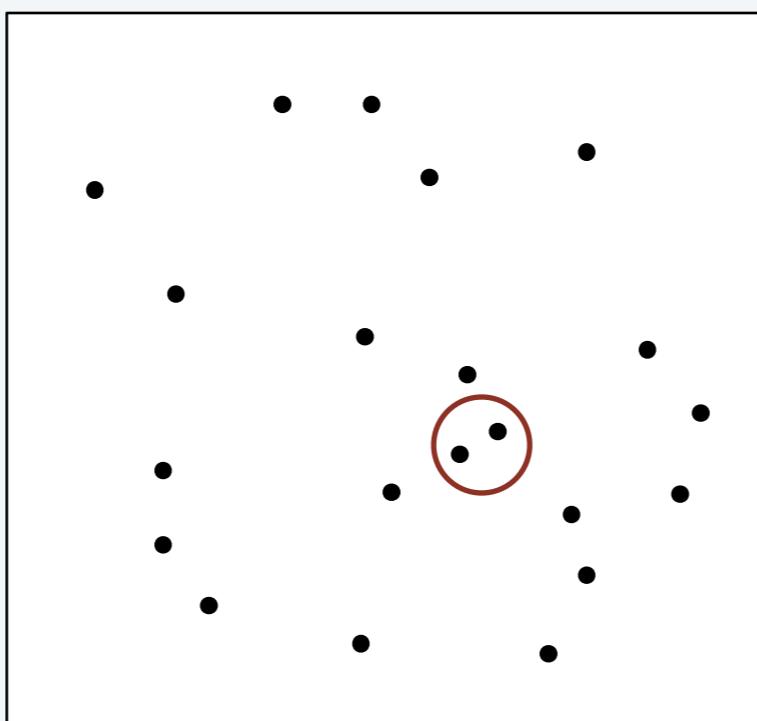
Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

Brute force. Check all pairs with $\Theta(n^2)$ distance calculations.

1D version. Easy $O(n \log n)$ algorithm if points are on a line.

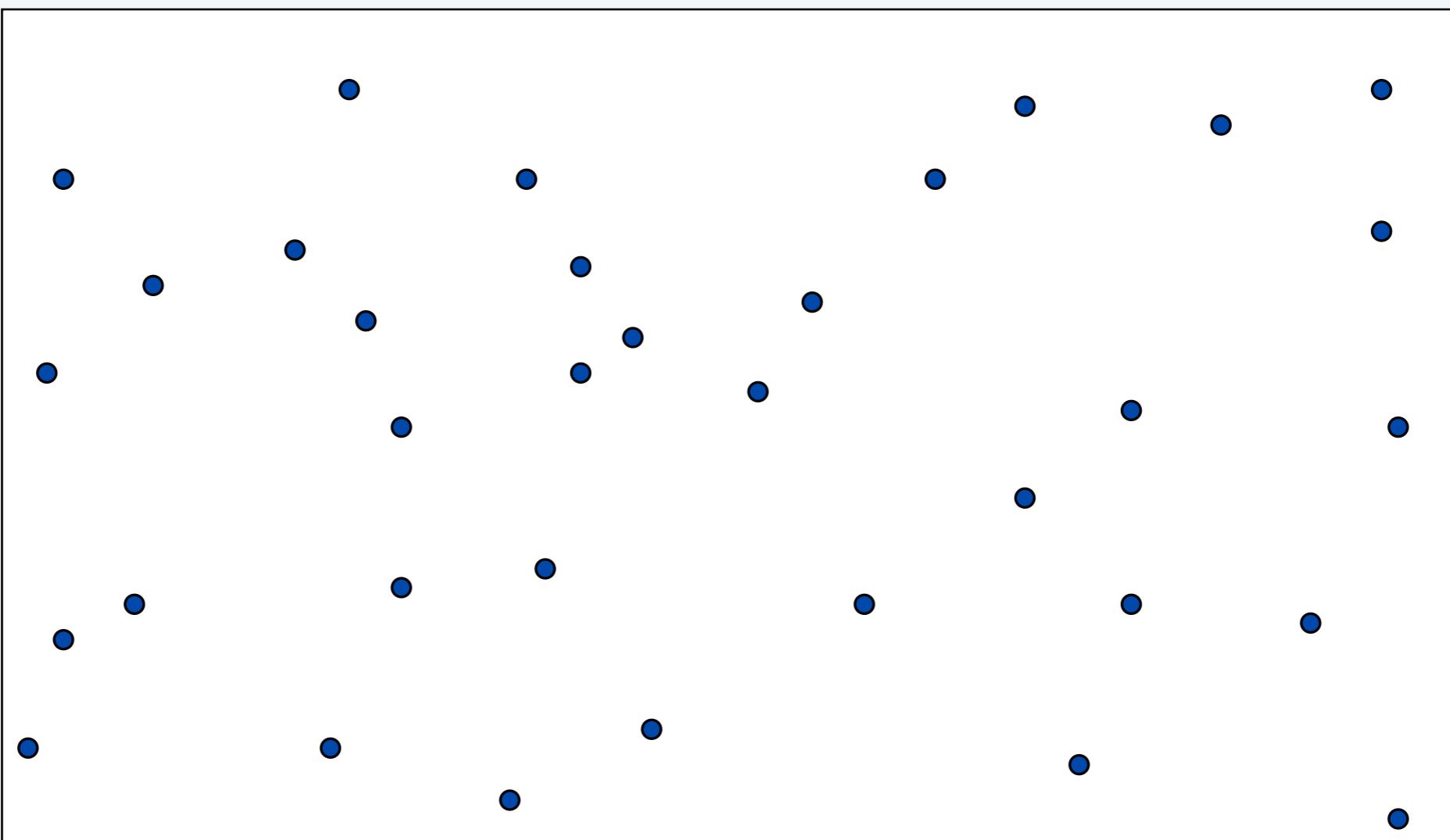
Non-degeneracy assumption. No two points have the same x -coordinate.



Closest pair of points: first attempt

Sorting solution.

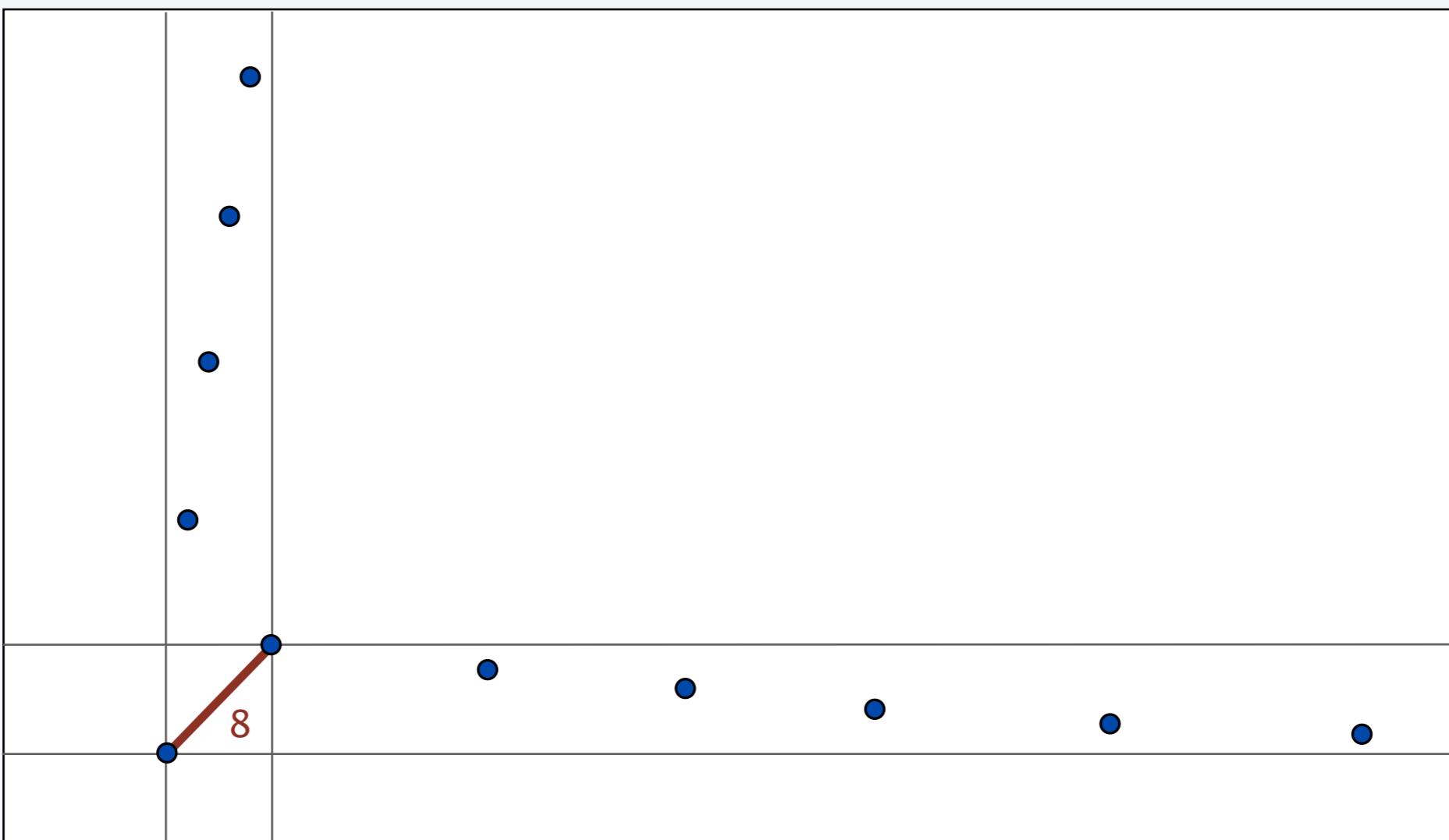
- Sort by x -coordinate and consider nearby points.
- Sort by y -coordinate and consider nearby points.



Closest pair of points: first attempt

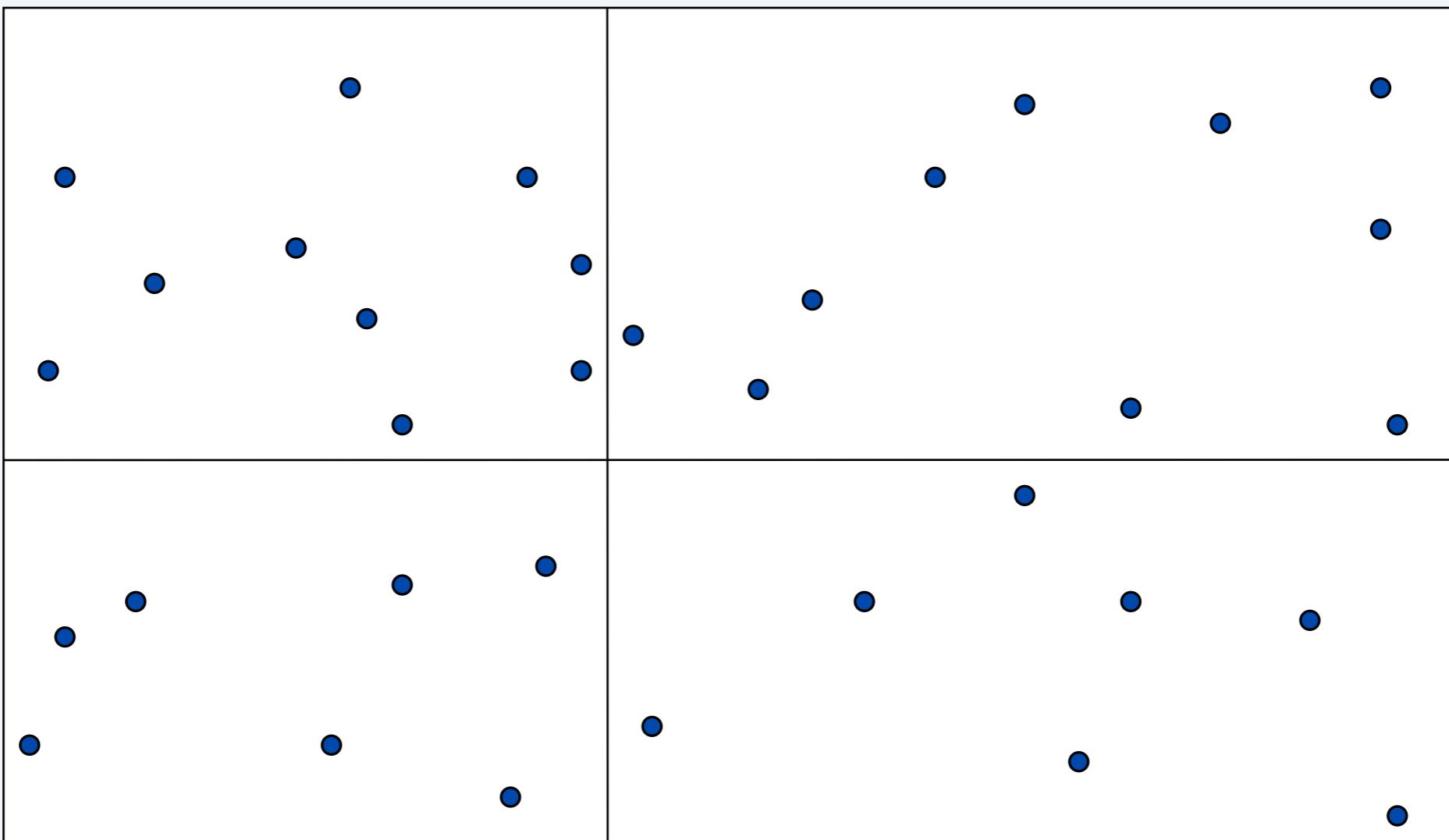
Sorting solution.

- Sort by x -coordinate and consider nearby points.
- Sort by y -coordinate and consider nearby points.



Closest pair of points: second attempt

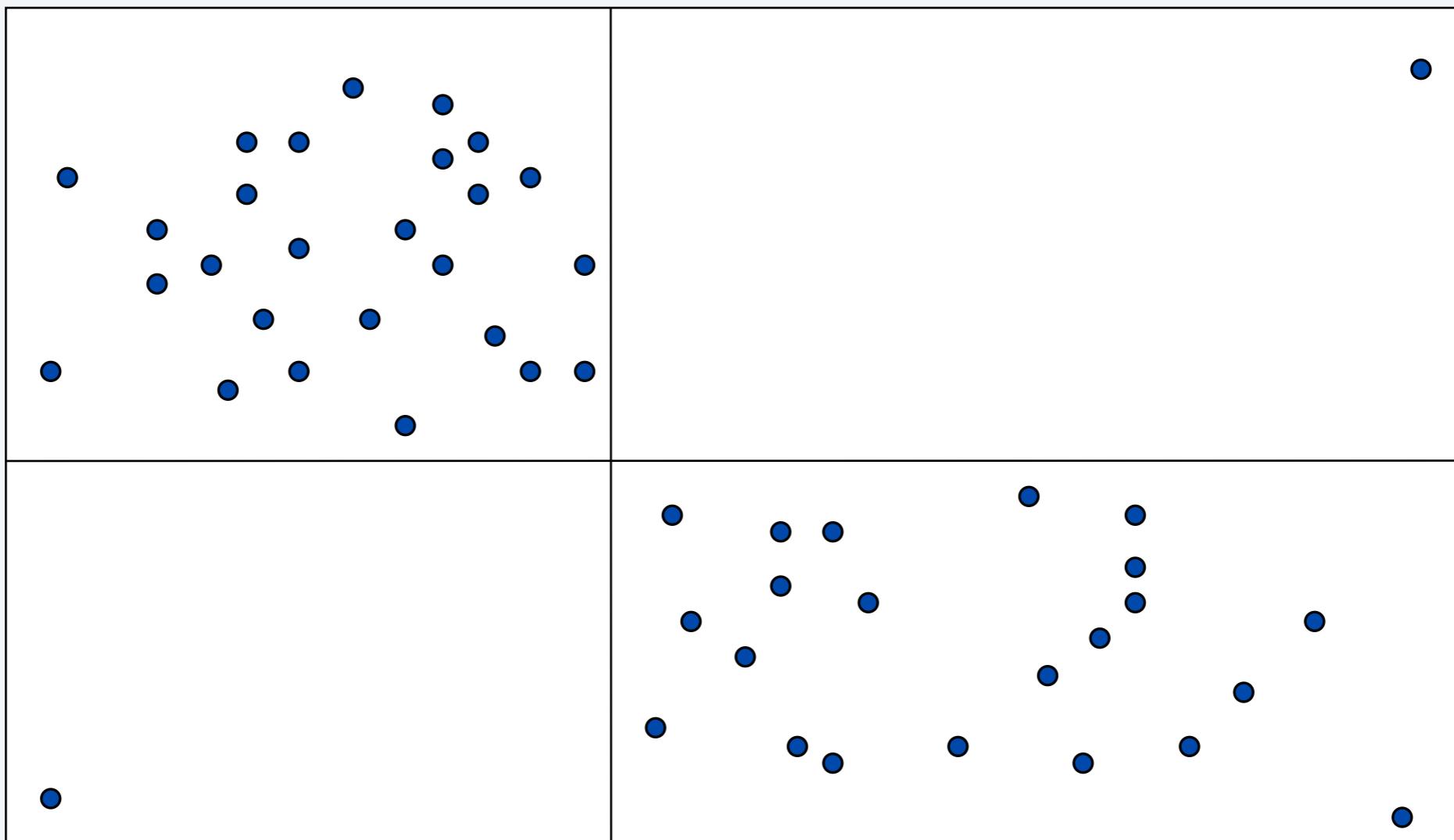
Divide. Subdivide region into 4 quadrants.



Closest pair of points: second attempt

Divide. Subdivide region into 4 quadrants.

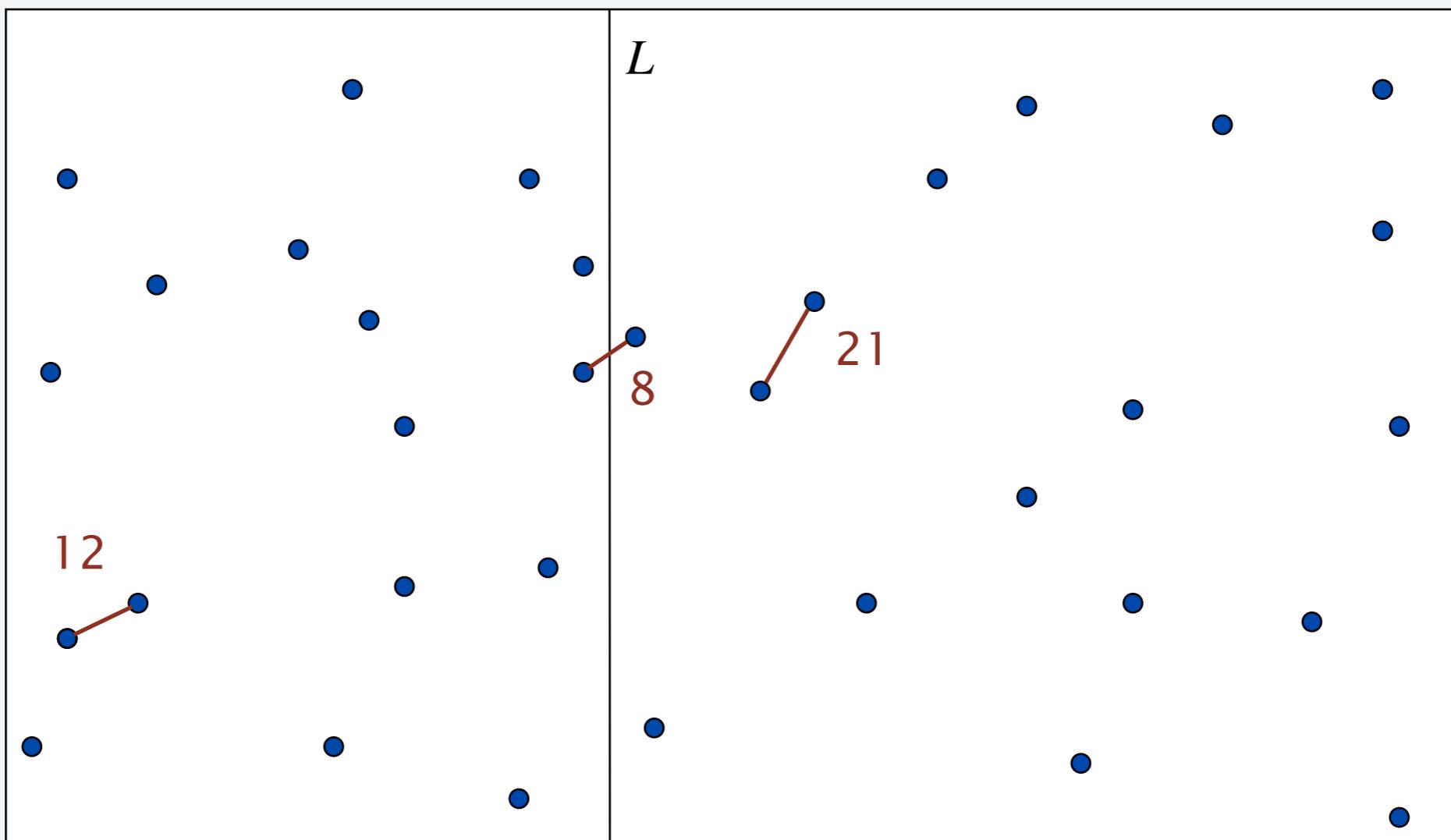
Obstacle. Impossible to ensure $n/4$ points in each piece.



Closest pair of points: divide-and-conquer algorithm

- Divide: draw vertical line L so that $n/2$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.

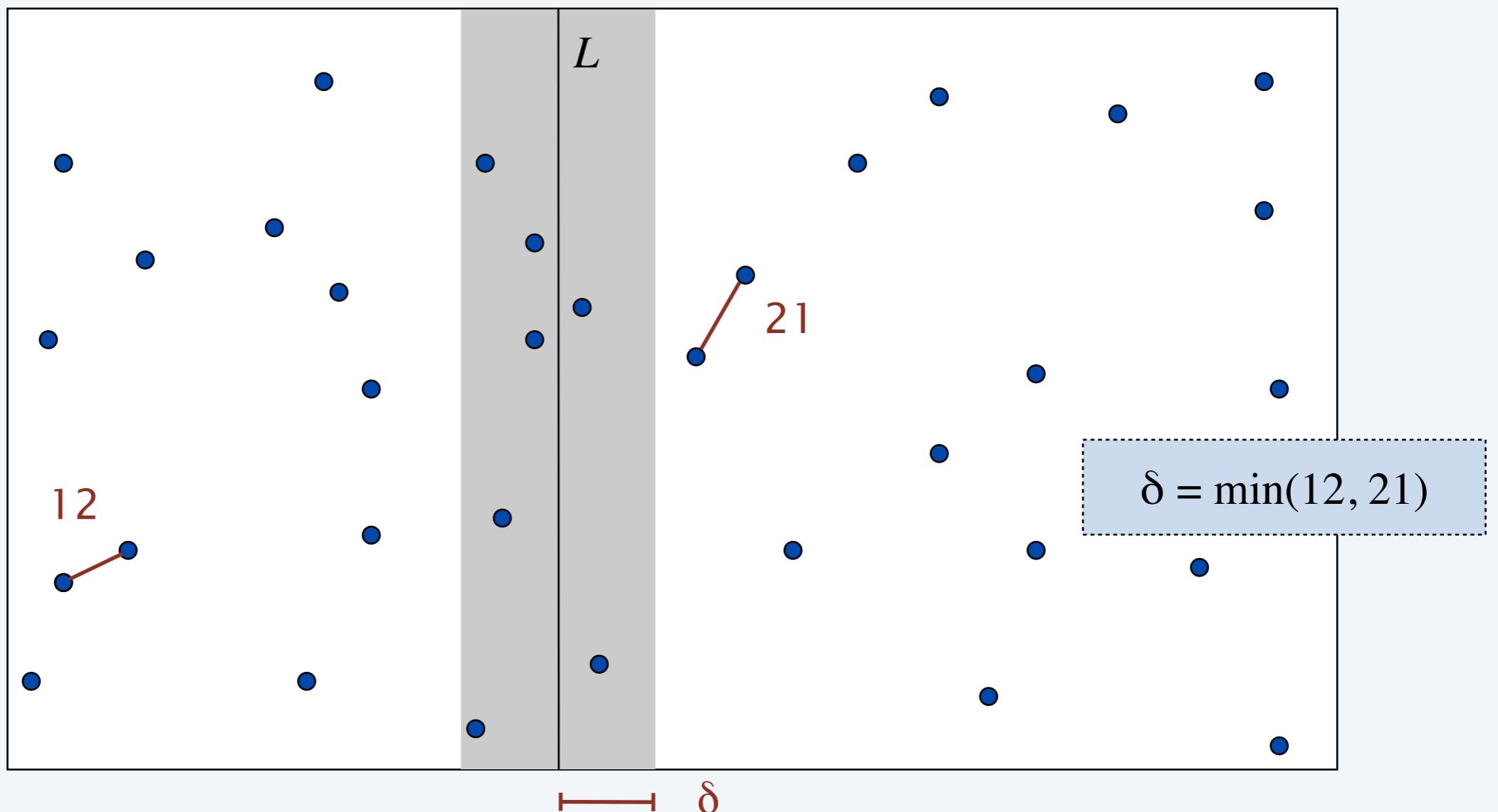
seems like $\Theta(n^2)$



How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance $< \delta$.

- Observation: suffices to consider only those points within δ of line L .

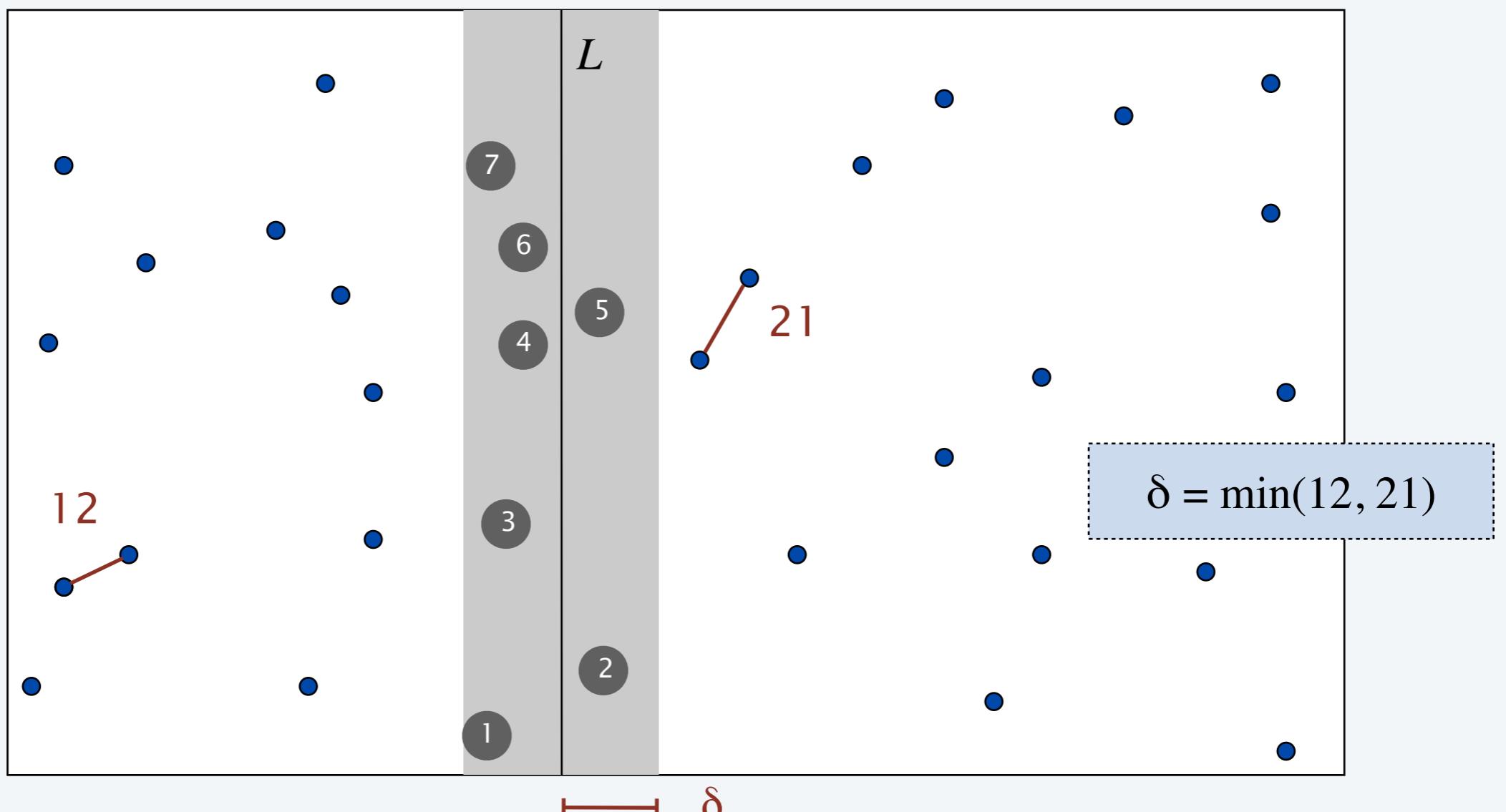


How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance $< \delta$.

- Observation: suffices to consider only those points within δ of line L .
- Sort points in 2δ -strip by their y -coordinate.
- Check distances of only those points within 7 positions in sorted list!

why?



How to find closest pair with one point in each side?

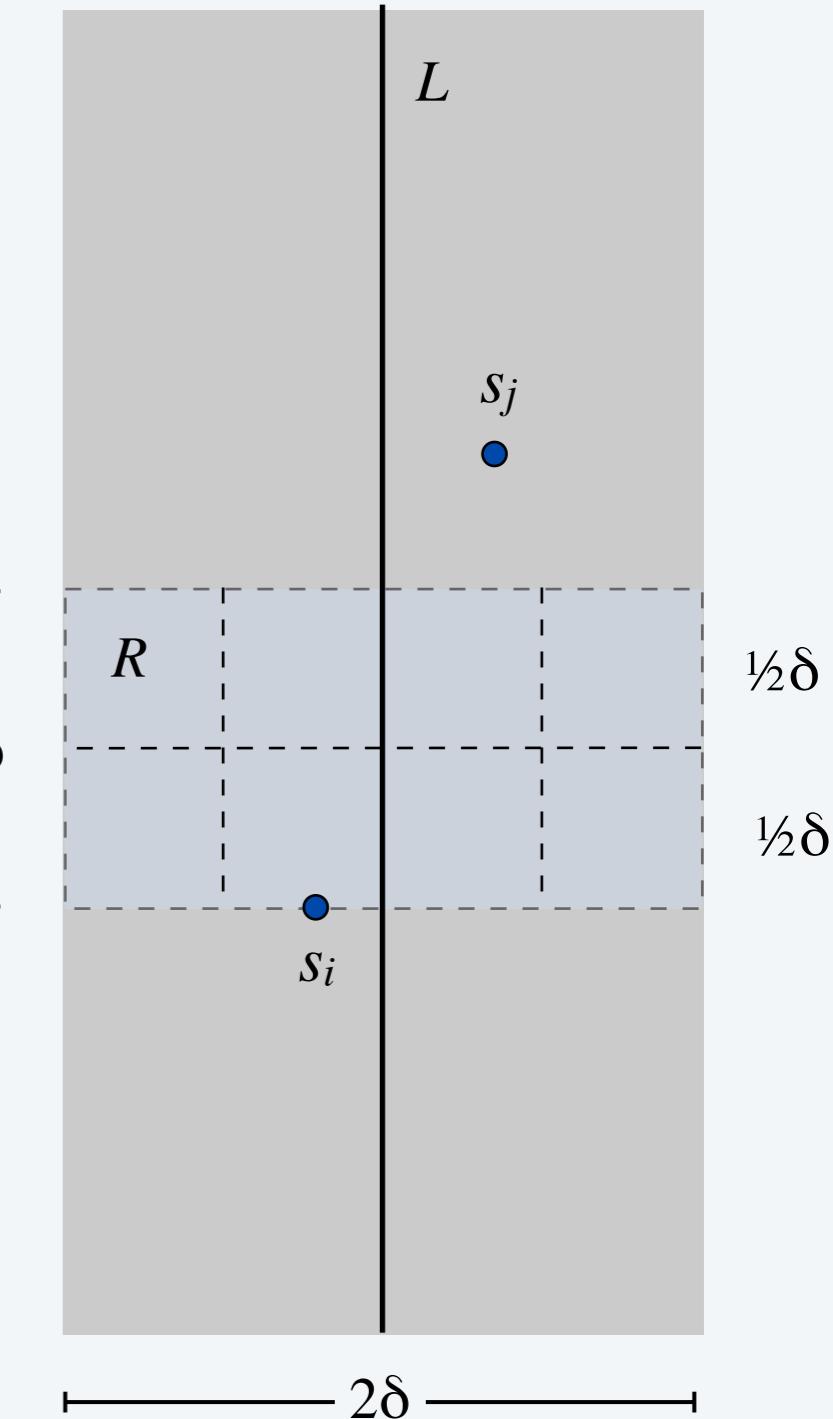
Def. Let s_i be the point in the 2δ -strip, with the i^{th} smallest y -coordinate.

Claim. If $|j - i| > 7$, then the distance between s_i and s_j is at least δ .

Pf.

- Consider the 2δ -by- δ rectangle R in strip whose min y -coordinate is y -coordinate of s_i .
- Distance between s_i and any point s_j above R is $\geq \delta$.
- Subdivide R into 8 squares.
diameter is $\delta / \sqrt{2} < \delta$
- At most 1 point per square.
- At most 7 other points can be in R . ■

constant can be improved with more refined geometric packing argument



Closest pair of points: divide-and-conquer algorithm

CLOSEST-PAIR(p_1, p_2, \dots, p_n)

Compute vertical line L such that half the points
are on each side of the line.

$\delta_1 \leftarrow \text{CLOSEST-PAIR}(\text{points in left half}).$

$\longleftarrow O(n)$

$\delta_2 \leftarrow \text{CLOSEST-PAIR}(\text{points in right half}).$

$\longleftarrow T(n / 2)$

$\delta \leftarrow \min \{ \delta_1, \delta_2 \}.$

$\longleftarrow T(n / 2)$

Delete all points further than δ from line L .

$\longleftarrow O(n)$

Sort remaining points by y -coordinate.

$\longleftarrow O(n \log n)$

Scan points in y -order and compare distance between
each point and next 7 neighbors. If any of these
distances is less than δ , update δ .

$\longleftarrow O(n)$

RETURN δ .



What is the solution to the following recurrence?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n \log n) & \text{if } n > 1 \end{cases}$$

- A. $T(n) = \Theta(n).$
- B. $T(n) = \Theta(n \log n).$
- C. $T(n) = \Theta(n \log^2 n).$
- D. $T(n) = \Theta(n^2).$

Refined version of closest-pair algorithm

Q. How to improve to $O(n \log n)$?

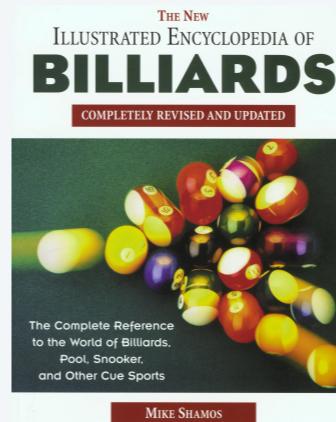
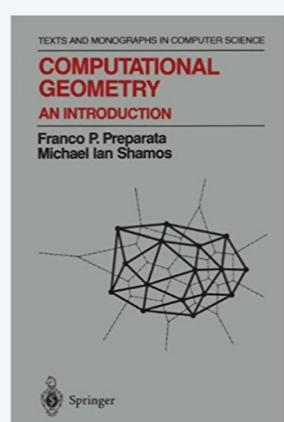
A. Don't sort points in strip from scratch each time.

- Each recursive call returns two lists: all points sorted by x -coordinate, and all points sorted by y -coordinate.
- Sort by **merging** two pre-sorted lists.

Theorem. [Shamos 1975] The divide-and-conquer algorithm for finding a closest pair of points in the plane can be implemented in $O(n \log n)$ time.

Pf.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$



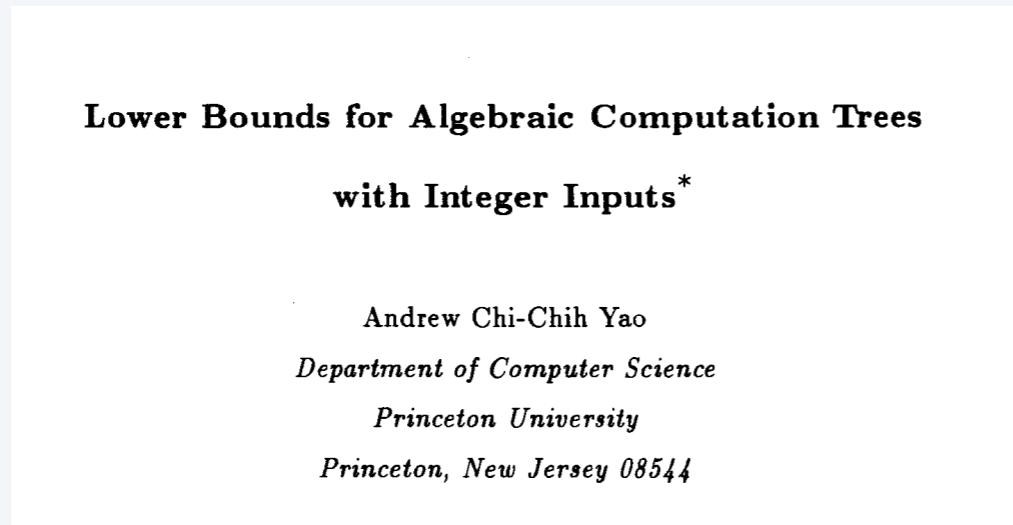


What is the complexity of the 2D closest pair problem?

- A. $\Theta(n)$.
- B. $\Theta(n \log^* n)$.
- C. $\Theta(n \log \log n)$.
- D. $\Theta(n \log n)$.
- E. Not even Tarjan knows.

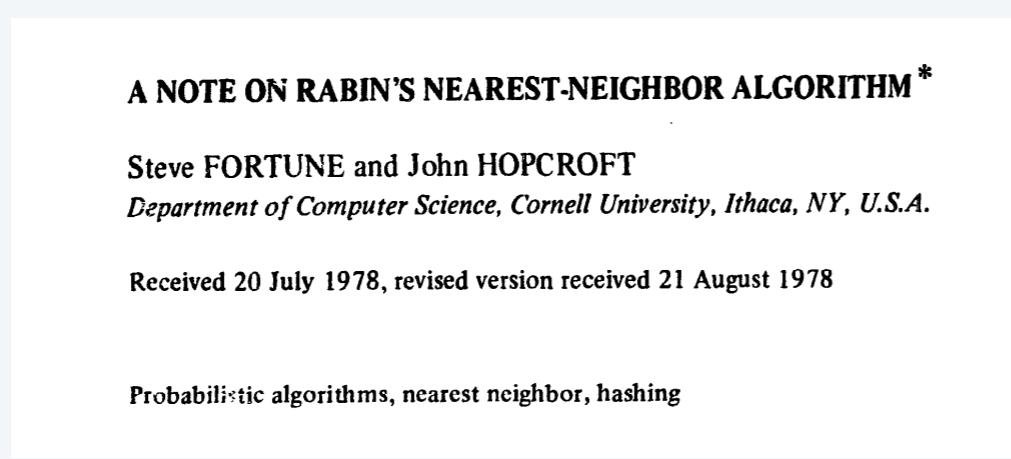
Computational complexity of closest-pair problem

Theorem. [Ben-Or 1983, Yao 1989] In quadratic decision tree model, any algorithm for closest pair (even in 1D) requires $\Omega(n \log n)$ quadratic tests.



$$(x_1 - x_2)^2 + (y_1 - y_2)^2$$

Theorem. [Rabin 1976] There exists an algorithm to find the closest pair of points in the plane whose expected running time is $O(n)$.



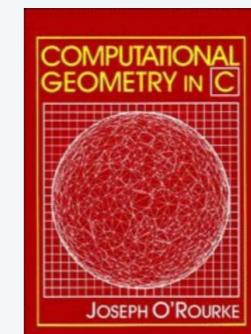
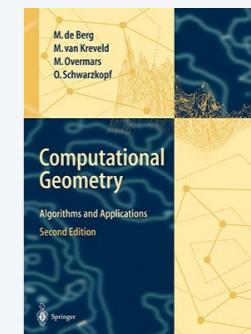
not subject to $\Omega(n \log n)$ lower bound
because it uses the floor function

Digression: computational geometry

Ingenious divide-and-conquer algorithms for core geometric problems.

problem	brute	clever
closest pair	$O(n^2)$	$O(n \log n)$
farthest pair	$O(n^2)$	$O(n \log n)$
convex hull	$O(n^2)$	$O(n \log n)$
Delaunay/Voronoi	$O(n^4)$	$O(n \log n)$
Euclidean MST	$O(n^2)$	$O(n \log n)$

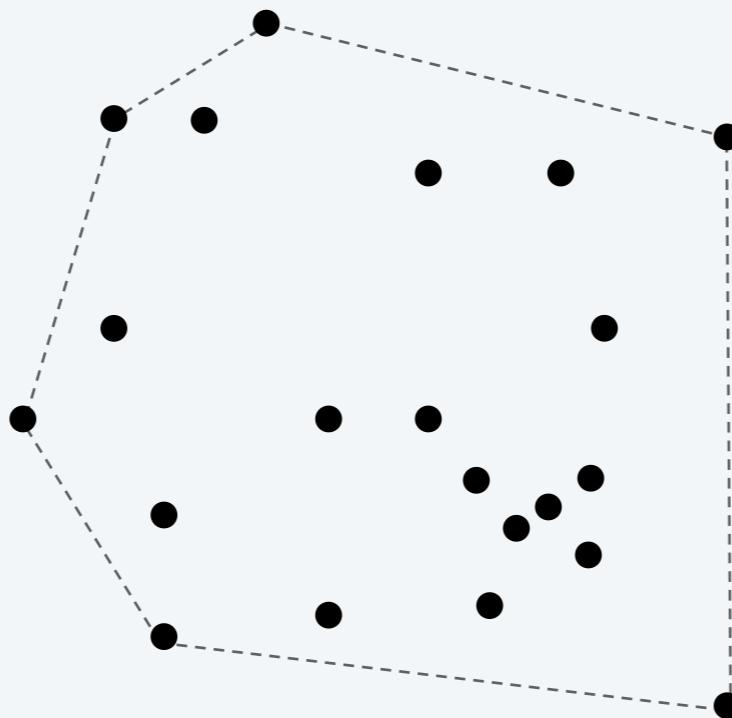
running time to solve a 2D problem with n points



Note. 3D and higher dimensions test limits of our ingenuity.

Convex hull

The **convex hull** of a set of n points is the smallest perimeter fence enclosing the points.

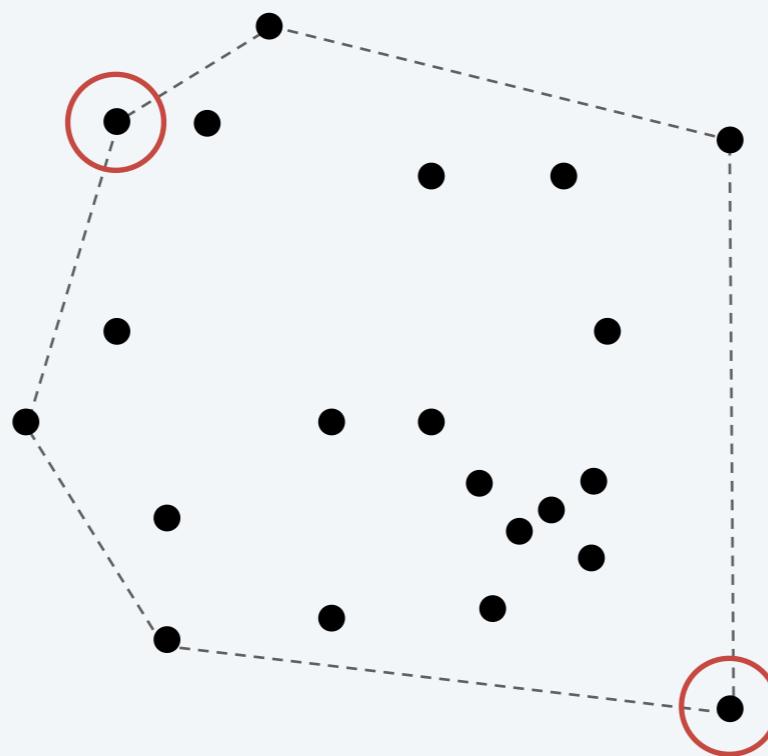


Equivalent definitions.

- Smallest area convex polygon enclosing the points.
- Intersection of all convex set containing all the points.

Farthest pair

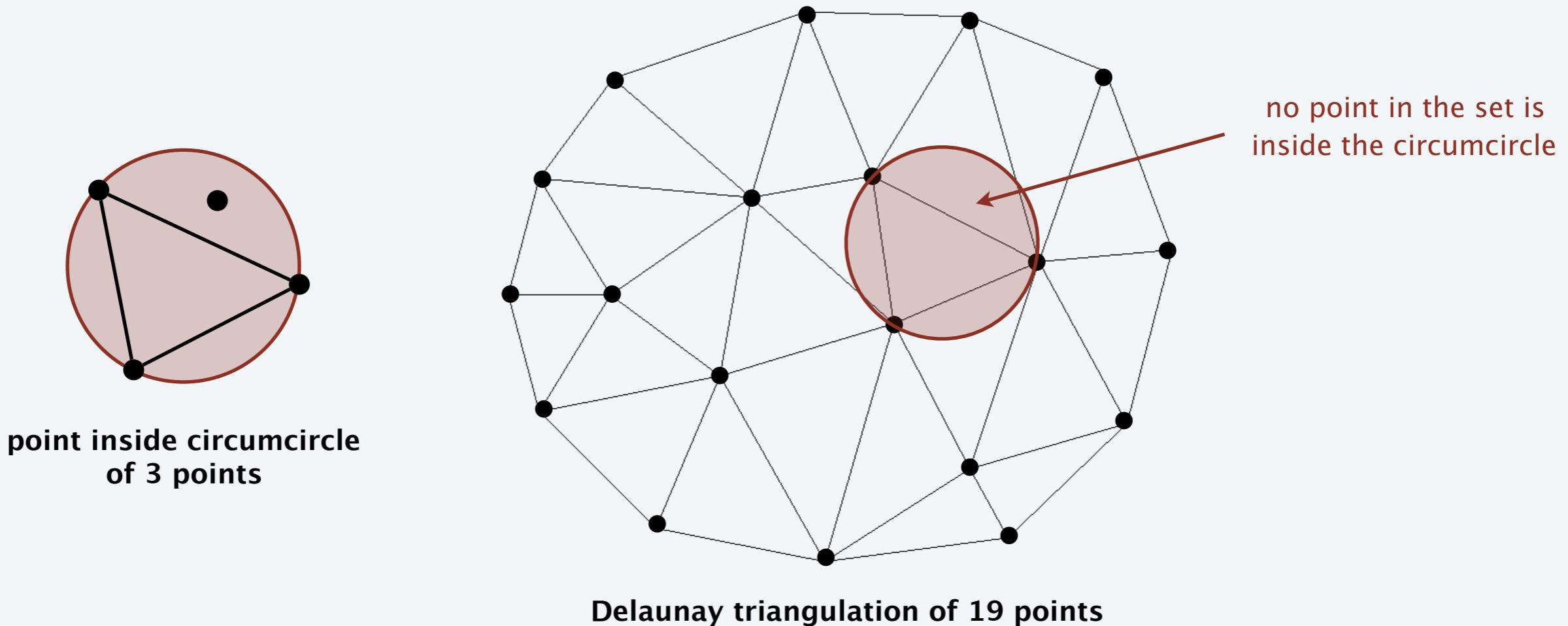
Given n points in the plane, find a pair of points with the largest Euclidean distance between them.



Fact. Points in farthest pair are extreme points on convex hull.

Delaunay triangulation

The **Delaunay triangulation** is a triangulation of n points in the plane such that no point is inside the circumcircle of any triangle.



Some useful properties.

- No edges cross.
- Among all triangulations, it maximizes the minimum angle.
- Contains an edge between each point and its nearest neighbor.

Euclidean MST

Given n points in the plane, find MST connecting them.

[distances between point pairs are Euclidean distances]



Fact. Euclidean MST is subgraph of Delaunay triangulation.

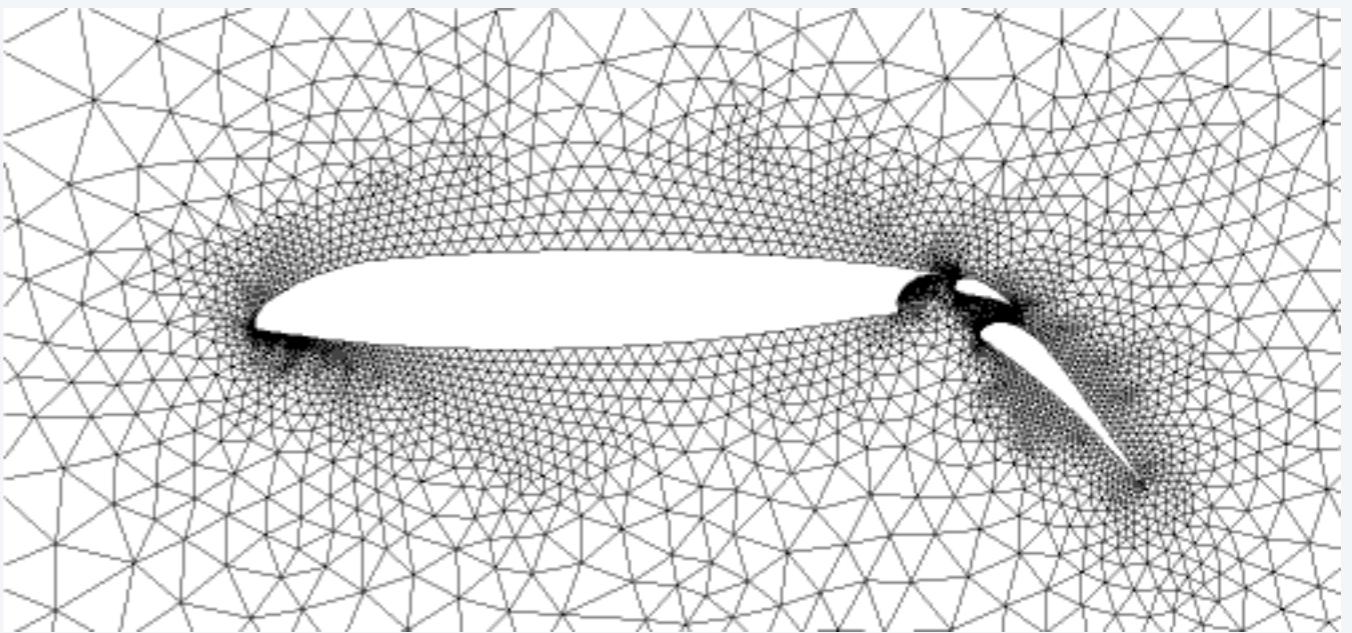
Implication. Can compute Euclidean MST in $O(n \log n)$ time.

- Compute Delaunay triangulation.
- Compute MST of Delaunay triangulation. ← it's planar
($\leq 3n$ edges)

Computational geometry applications

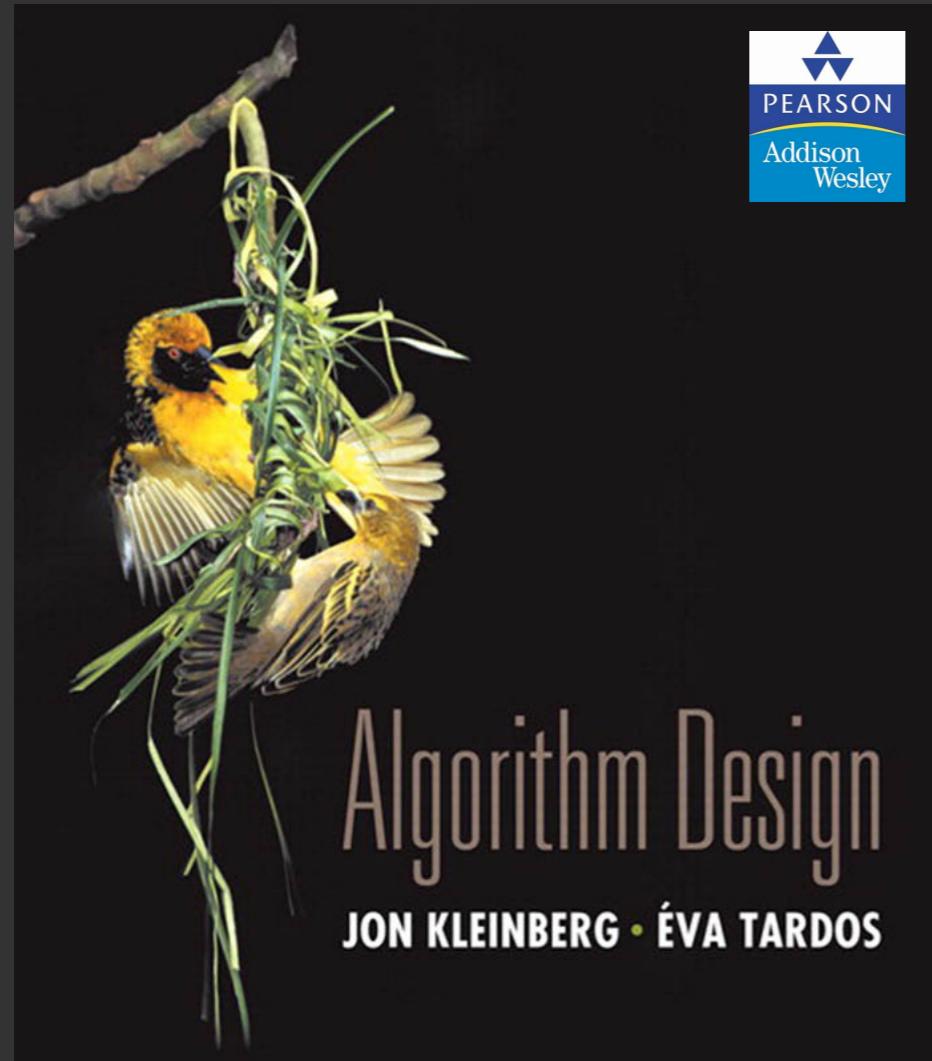
Applications.

- Robotics.
- VLSI design.
- Data mining.
- Medical imaging.
- Computer vision.
- Scientific computing.
- Finite-element meshing.
- Astronomical simulation.
- Models of physical world.
- Geographic information systems.
- Computer graphics (movies, games, virtual reality).



airflow around an aircraft wing

<http://www.ics.uci.edu/~eppstein/geom.html>



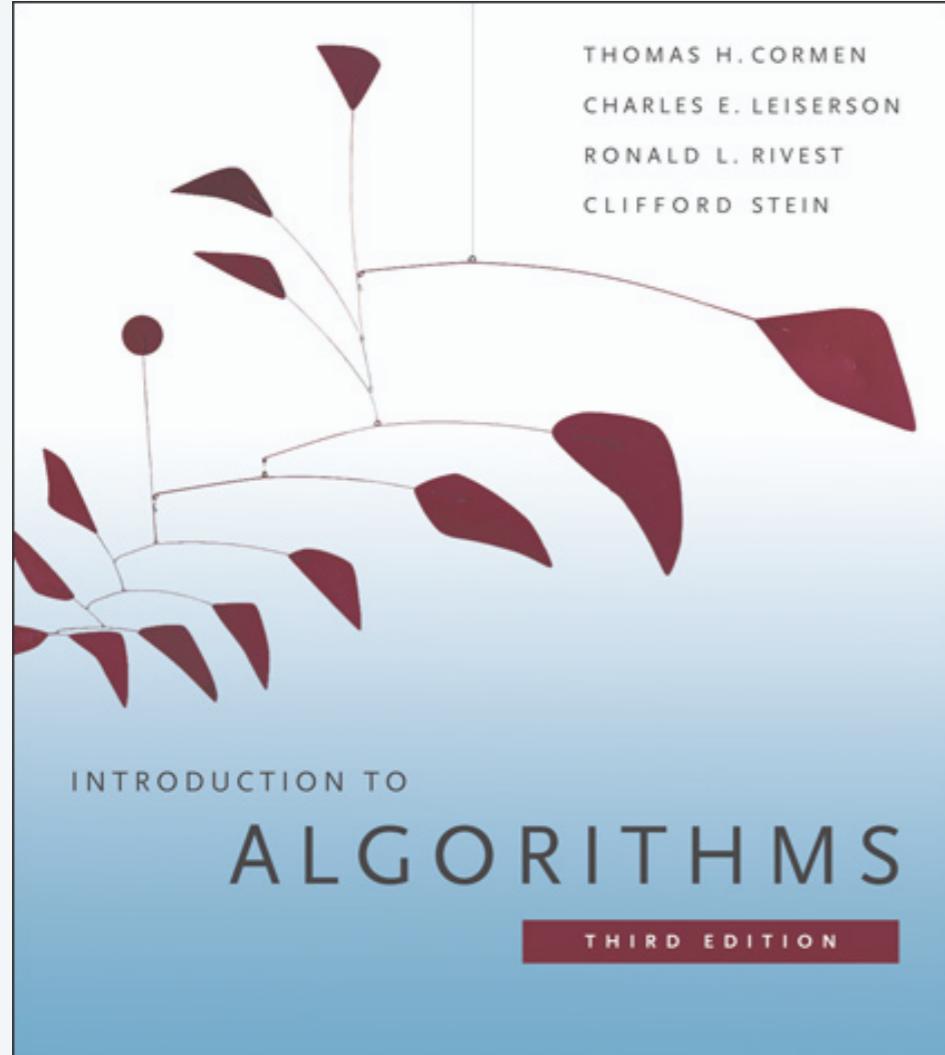
DIVIDE AND CONQUER II

- ▶ *master theorem*
- ▶ *integer multiplication*
- ▶ *matrix multiplication*
- ▶ *convolution and FFT*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>



SECTIONS 4.4-4.6

DIVIDE AND CONQUER II

- ▶ *master theorem*
- ▶ *integer multiplication*
- ▶ *matrix multiplication*
- ▶ *convolution and FFT*

Divide-and-conquer recurrences

Goal. Recipe for solving common divide-and-conquer recurrences:

$$T(n) = a T\left(\frac{n}{b}\right) + f(n)$$

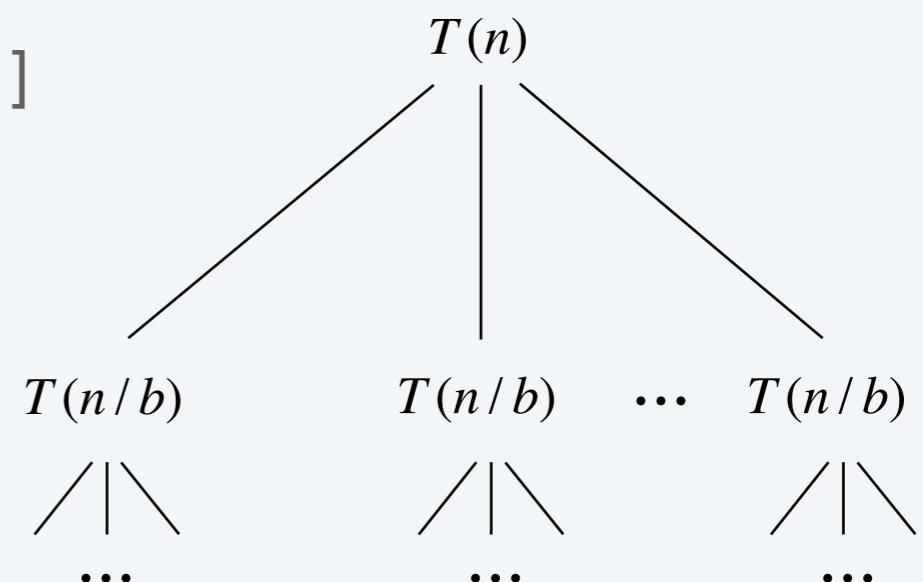
with $T(0) = 0$ and $T(1) = \Theta(1)$.

Terms.

- $a \geq 1$ is the number of subproblems.
- $b \geq 2$ is the factor by which the subproblem size decreases.
- $f(n) \geq 0$ is the work to divide and combine subproblems.

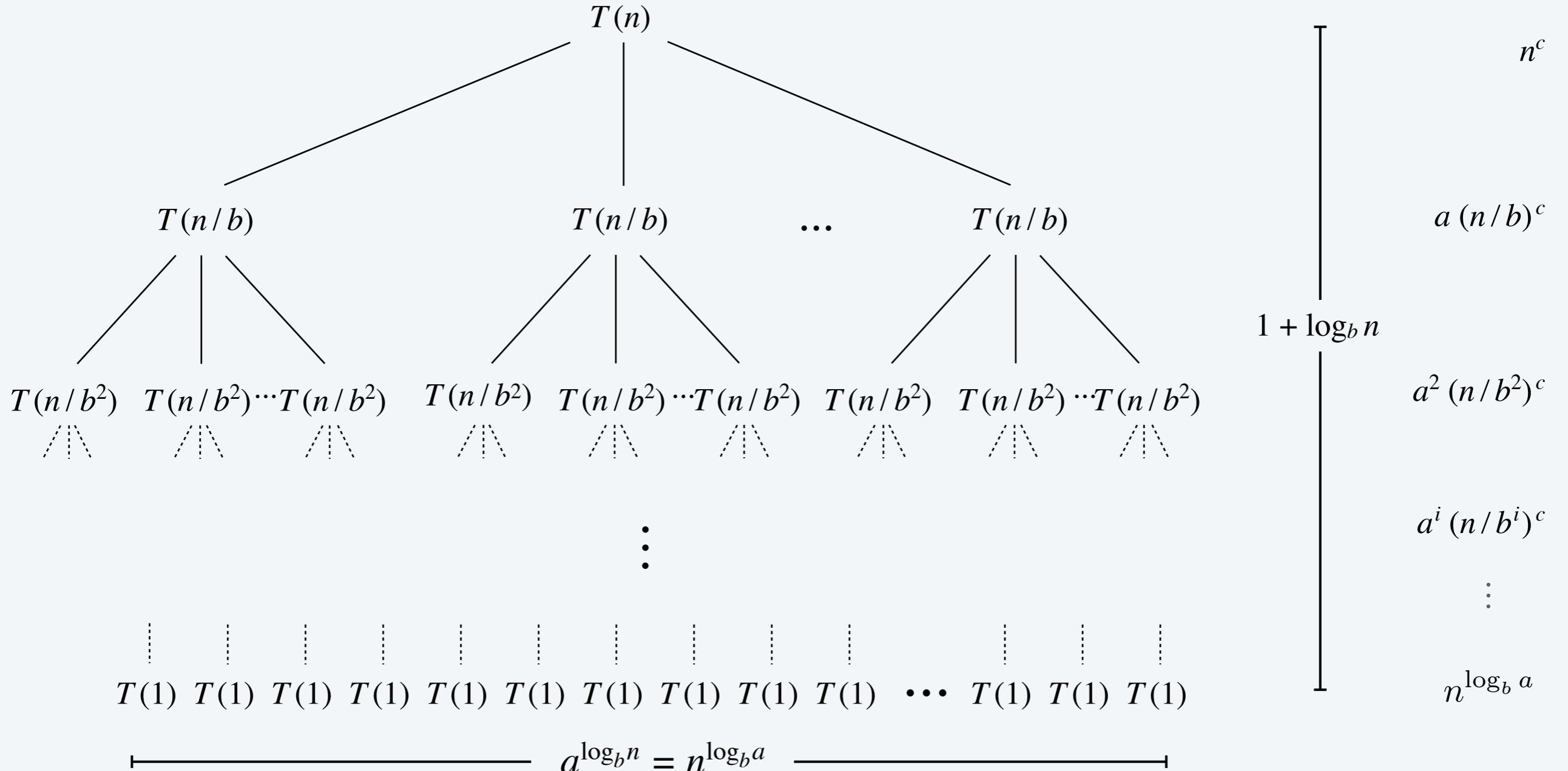
Recursion tree. [assuming n is a power of b]

- a = branching factor.
- a^i = number of subproblems at level i .
- $1 + \log_b n$ levels.
- n / b^i = size of subproblem at level i .



Divide-and-conquer recurrences: recursion tree

Suppose $T(n)$ satisfies $T(n) = a T(n / b) + n^c$ with $T(1) = 1$, for n a power of b .



$$r = a / b^c \quad T(n) = n^c \sum_{i=0}^{\log_b n} r^i$$

Divide-and-conquer recurrences: recursion tree analysis

Suppose $T(n)$ satisfies $T(n) = a T(n / b) + n^c$ with $T(1) = 1$, for n a power of b .

Let $r = a / b^c$. Note that $r < 1$ iff $c > \log_b a$.

$$T(n) = n^c \sum_{i=0}^{\log_b n} r^i = \begin{cases} \Theta(n^c) & \text{if } r < 1 \\ \Theta(n^c \log n) & \text{if } r = 1 \\ \Theta(n^{\log_b a}) & \text{if } r > 1 \end{cases}$$

$c > \log_b a \quad \leftarrow$ cost dominated by cost of root
 $c = \log_b a \quad \leftarrow$ cost evenly distributed in tree
 $c < \log_b a \quad \leftarrow$ cost dominated by cost of leaves

Geometric series.

- If $0 < r < 1$, then $1 + r + r^2 + r^3 + \dots + r^k \leq 1 / (1 - r)$.
- If $r = 1$, then $1 + r + r^2 + r^3 + \dots + r^k = k + 1$.
- If $r > 1$, then $1 + r + r^2 + r^3 + \dots + r^k = (r^{k+1} - 1) / (r - 1)$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a T\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.



Pf sketch.

- Prove when b is an integer and n is an exact power of b .
- Extend domain of recurrences to reals (or rationals).
- Deal with floors and ceilings. ← at most 2 extra levels in recursion tree

$$\begin{aligned} \lceil \lceil \lceil n/b \rceil / b \rceil / b \rceil &< n/b^3 + (1/b^2 + 1/b + 1) \\ &\leq n/b^3 + 2 \end{aligned}$$

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a T\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.



Extensions.

- Can replace Θ with O everywhere.
- Can replace Θ with Ω everywhere.
- Can replace initial conditions with $T(n) = \Theta(1)$ for all $n \leq n_0$ and require recurrence to hold only for all $n > n_0$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a T\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.



Ex. [Case 1] $T(n) = 3 T(\lfloor n/2 \rfloor) + 5n$.

- $a = 3$, $b = 2$, $c = 1 < \log_b a = 1.5849\dots$
- $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.58})$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a T\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

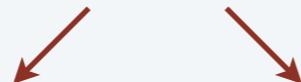
Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.



ok to intermix floor and ceiling



Ex. [Case 2] $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 17n$.

- $a = 2$, $b = 2$, $c = 1 = \log_b a$.
- $T(n) = \Theta(n \log n)$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a T\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.



Ex. [Case 3] $T(n) = 48 T(\lfloor n/4 \rfloor) + n^3$.

- $a = 48$, $b = 4$, $c = 3 > \log_b a = 2.7924\dots$
- $T(n) = \Theta(n^3)$.

Master theorem need not apply

Gaps in master theorem.

- Number of subproblems is not a constant.

$$T(n) = \textcircled{n} T(n/2) + n^2$$

- Number of subproblems is less than 1.

$$T(n) = \textcircled{\frac{1}{2}} T(n/2) + n^2$$

- Work to divide and combine subproblems is not $\Theta(n^c)$.

$$T(n) = 2 T(n/2) + \textcircled{n \log n}$$



Consider the following recurrence. Which case of the master theorem?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 3T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

- A. Case 3: $T(n) = \Theta(n)$.
- B. Case 2: $T(n) = \Theta(n \log n)$.
- C. Case 1: $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$.
- D. Master theorem not applicable.



Consider the following recurrence. Which case of the master theorem?

$$T(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n & \text{if } n > 1 \end{cases}$$

- A. Case 1: $T(n) = \Theta(n)$.
- B. Case 2: $T(n) = \Theta(n \log n)$.
- C. Case 3: $T(n) = \Theta(n)$.
- D. Master theorem not applicable.

Akra–Bazzi theorem

Theorem. [Akra–Bazzi 1998] Given constants $a_i > 0$ and $0 < b_i < 1$, functions $|h_i(n)| = O(n / \log^2 n)$ and $g(n) = O(n^c)$. If $T(n)$ satisfies the recurrence:

$$T(n) = \sum_{i=1}^k a_i T(b_i n + h_i(n)) + g(n)$$

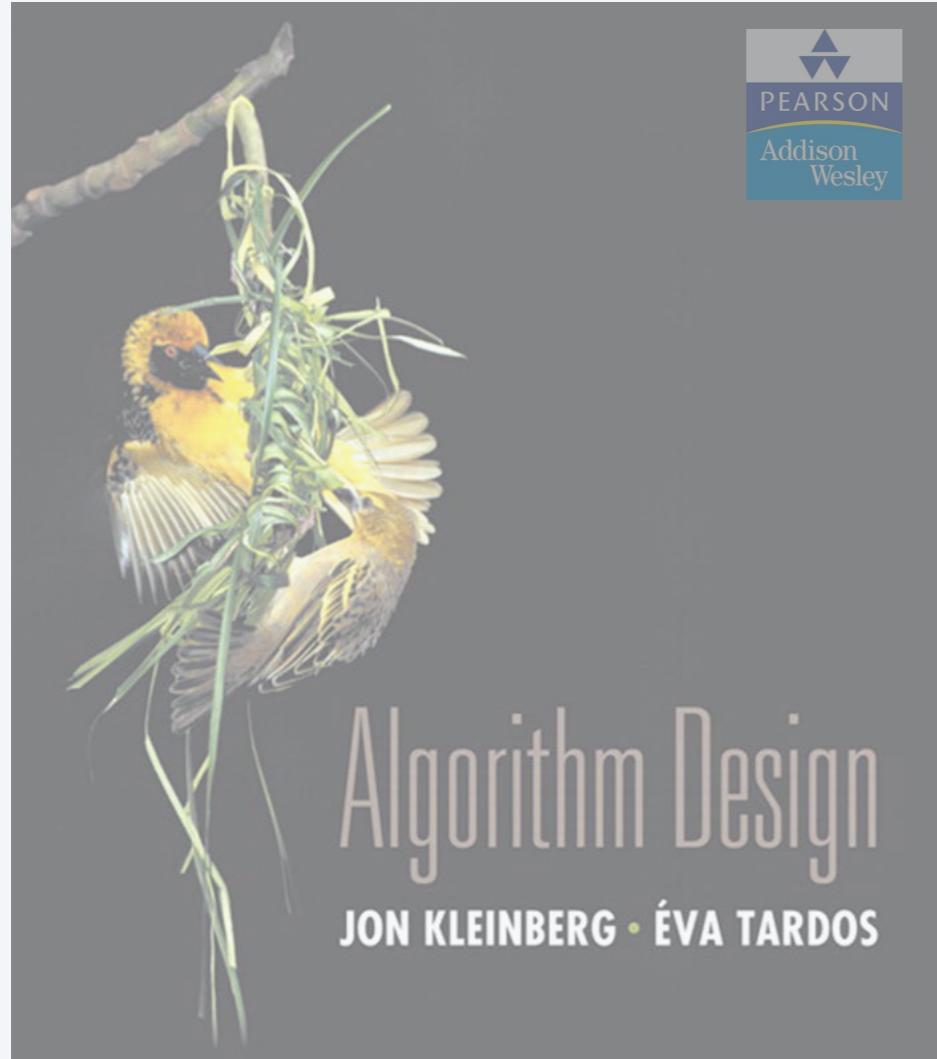
↑
 a_i subproblems
of size $b_i n$ ↑
small perturbation to handle
floors and ceilings



then, $T(n) = \Theta\left(n^p \left(1 + \int_1^n \frac{g(u)}{u^{p+1}} du\right)\right)$, where p satisfies $\sum_{i=1}^k a_i b_i^p = 1$.

Ex. $T(n) = T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5 n$, with $T(0) = 0$ and $T(1) = 0$.

- $a_1 = 1, b_1 = 1/5, a_2 = 1, b_2 = 7/10 \Rightarrow p = 0.83978\dots < 1$.
- $h_1(n) = \lfloor n/5 \rfloor - n/5, h_2(n) = 3/10 n - 3\lfloor n/10 \rfloor$.
- $g(n) = 11/5 n \Rightarrow T(n) = \Theta(n)$.



SECTION 5.5

DIVIDE AND CONQUER II

- ▶ *master theorem*
- ▶ *integer multiplication*
- ▶ *matrix multiplication*
- ▶ *convolution and FFT*

Integer addition and subtraction

Addition. Given two n -bit integers a and b , compute $a + b$.

Subtraction. Given two n -bit integers a and b , compute $a - b$.

Grade-school algorithm. $\Theta(n)$ bit operations. ← “bit complexity”
(instead of word RAM)

1	1	1	1	1	1	0	1
	1	1	0	1	0	1	0
+	0	1	1	1	1	1	0
	1	0	1	0	1	0	0
	1	0	1	0	1	0	0

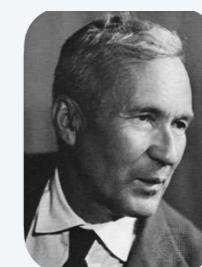
Remark. Grade-school addition and subtraction algorithms are optimal.

Integer multiplication

Multiplication. Given two n -bit integers a and b , compute $a \times b$.

Grade-school algorithm (long multiplication). $\Theta(n^2)$ bit operations.

	1	1	0	1	0	1	0	1
\times	0	1	1	1	1	1	0	1
	1	1	0	1	0	1	0	1
	0	0	0	0	0	0	0	0
	1	1	0	1	0	1	0	1
	1	1	0	1	0	1	0	1
	1	1	0	1	0	1	0	1
	1	1	0	1	0	1	0	1
	0	0	0	0	0	0	0	0
	0	1	1	0	1	0	0	0



Conjecture. [Kolmogorov 1956] Grade-school algorithm is optimal.

Theorem. [Karatsuba 1960] Conjecture is false.

Divide-and-conquer multiplication

To multiply two n -bit integers x and y :

- Divide x and y into low- and high-order bits.
- Multiply **four** $\frac{1}{2}n$ -bit integers, recursively.
- Add and shift to obtain result.

$$m = \lceil n / 2 \rceil$$

$$\begin{array}{ll} a = \lfloor x / 2^m \rfloor & b = x \bmod 2^m \\ c = \lfloor y / 2^m \rfloor & d = y \bmod 2^m \end{array}$$

← use bit shifting
to compute 4 terms

$$x y = (2^m a + b) (2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

1 2 3 4

Ex. $x = 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1$ $y = 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1$

$\underbrace{}_a$ $\underbrace{}_b$ $\underbrace{}_c$ $\underbrace{}_d$

Divide-and-conquer multiplication

MULTIPLY(x, y, n)

IF ($n = 1$)

RETURN $x \times y$.

ELSE

$m \leftarrow \lceil n / 2 \rceil$.

$a \leftarrow \lfloor x / 2^m \rfloor; b \leftarrow x \bmod 2^m$.

$c \leftarrow \lfloor y / 2^m \rfloor; d \leftarrow y \bmod 2^m$.

$e \leftarrow \text{MULTIPLY}(a, c, m)$.

$f \leftarrow \text{MULTIPLY}(b, d, m)$.

$g \leftarrow \text{MULTIPLY}(b, c, m)$.

$h \leftarrow \text{MULTIPLY}(a, d, m)$.

RETURN $2^{2m} e + 2^m (g + h) + f$. $\longleftarrow \Theta(n)$

$\Theta(n)$

$4 T(\lceil n / 2 \rceil)$



How many bit operations to multiply two n -bit integers using the divide-and-conquer multiplication algorithm?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 4T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

- A.** $T(n) = \Theta(n^{1/2})$.
- B.** $T(n) = \Theta(n \log n)$.
- C.** $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$.
- D.** $T(n) = \Theta(n^2)$.

Karatsuba trick

To multiply two n -bit integers x and y :

- Divide x and y into low- and high-order bits.
- To compute middle term $bc + ad$, use identity:

$$bc + ad = ac + bd - (a - b)(c - d)$$

- Multiply only three $\frac{1}{2}n$ -bit integers, recursively.

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

$$x y = (2^m a + b) (2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

$$= 2^{2m} ac + 2^m (ac + bd - (a - b)(c - d)) + bd$$

1

1

3

2

3

$$x = 1 \overbrace{000}^a \overbrace{1101}^b 1$$

$$y = 1 \overbrace{110}^c \overbrace{0001}^d 1$$

Karatsuba multiplication

KARATSUBA-MULTIPLY(x, y, n)

IF ($n = 1$)

RETURN $x \times y$.

ELSE

$m \leftarrow \lceil n / 2 \rceil$.

$a \leftarrow \lfloor x / 2^m \rfloor; b \leftarrow x \bmod 2^m$.

$c \leftarrow \lfloor y / 2^m \rfloor; d \leftarrow y \bmod 2^m$.

$e \leftarrow \text{KARATSUBA-MULTIPLY}(a, c, m)$.

$f \leftarrow \text{KARATSUBA-MULTIPLY}(b, d, m)$.

$g \leftarrow \text{KARATSUBA-MULTIPLY}(|a - b|, |c - d|, m)$.

Flip sign of g if needed.

RETURN $2^{2m} e + 2^m (e + f - g) + f$. ————— $\Theta(n)$

————— $\Theta(n)$

————— $3 T(\lceil n / 2 \rceil)$

Karatsuba analysis

Proposition. Karatsuba's algorithm requires $O(n^{1.585})$ bit operations to multiply two n -bit integers.

Pf. Apply Case 3 of the master theorem to the recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 3T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$$\implies T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$$

Practice.

- Use base 32 or 64 (instead of base 2).
- Faster than grade-school algorithm for about 320–640 bits.

Integer arithmetic reductions

Integer multiplication. Given two n -bit integers, compute their product.

arithmetic problem	formula	bit complexity
integer multiplication	$a \times b$	$M(n)$
integer square	a^2	$\Theta(M(n))$
integer division	$\lfloor a / b \rfloor, a \bmod b$	$\Theta(M(n))$
integer square root	$\lfloor \sqrt{a} \rfloor$	$\Theta(M(n))$

integer arithmetic problems with the same bit complexity $M(n)$ as integer multiplication

$$ab = \frac{(a+b)^2 - a^2 - b^2}{2}$$



History of asymptotic complexity of integer multiplication

year	algorithm	bit operations
12xx	grade school	$O(n^2)$
1962	Karatsuba-Ofman	$O(n^{1.585})$
1963	Toom-3, Toom-4	$O(n^{1.465}), O(n^{1.404})$
1966	Toom-Cook	$O(n^{1+\varepsilon})$
1971	Schönhage-Strassen	$O(n \log n \cdot \log \log n)$
2007	Fürer	$n \log n 2^{O(\log^* n)}$
2019	Harvey-van der Hoeven	$O(n \log n)$
	???	$O(n)$

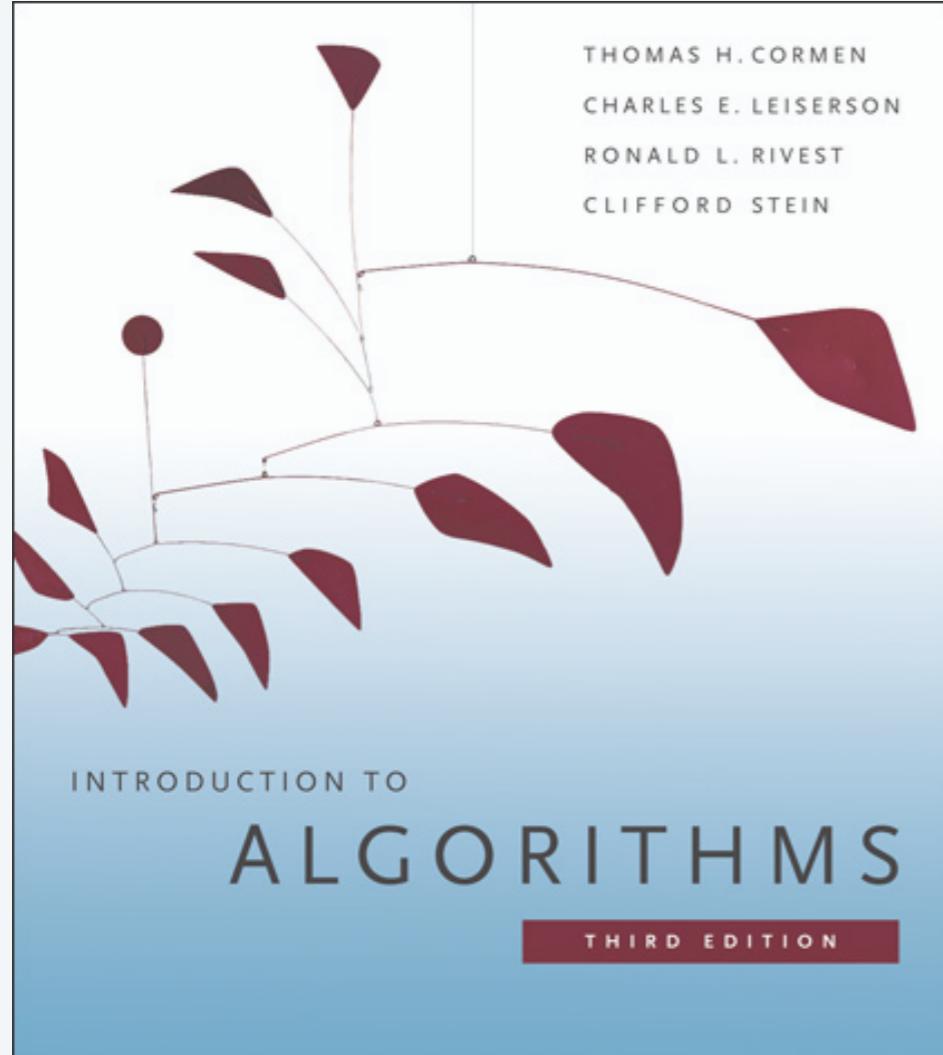
number of bit operations to multiply two n -bit integers

Remark. GNU Multiple Precision library uses one of first five algorithms depending on n .



used in Maple, Mathematica, gcc, cryptography, ...

GMP
«Arithmetic without limitations»



SECTION 4.2

DIVIDE AND CONQUER II

- ▶ *master theorem*
- ▶ *integer multiplication*
- ▶ *matrix multiplication*
- ▶ *convolution and FFT*

Dot product

Dot product. Given two length- n vectors a and b , compute $c = a \cdot b$.

Grade-school. $\Theta(n)$ arithmetic operations.

$$a \cdot b = \sum_{i=1}^n a_i b_i$$

$$a = [.70 \quad .20 \quad .10]$$

$$b = [.30 \quad .40 \quad .30]$$

$$a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32$$

Remark. “Grade-school” dot product algorithm is asymptotically optimal.

Matrix multiplication

Matrix multiplication. Given two n -by- n matrices A and B , compute $C = AB$.

Grade-school. $\Theta(n^3)$ arithmetic operations.

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

Q. Is “grade-school” matrix multiplication algorithm asymptotically optimal?

Block matrix multiplication

$$\begin{matrix}
 & C_{11} \\
 & \searrow \\
 \left[\begin{array}{cccc}
 152 & 158 & 164 & 170 \\
 504 & 526 & 548 & 570 \\
 856 & 894 & 932 & 970 \\
 1208 & 1262 & 1316 & 1370
 \end{array} \right] & = & \left[\begin{array}{cc|cc}
 0 & 1 & 2 & 3 \\
 4 & 5 & 6 & 7 \\
 \hline
 8 & 9 & 10 & 11 \\
 12 & 13 & 14 & 15
 \end{array} \right] & \times & \left[\begin{array}{cccc}
 16 & 17 & 18 & 19 \\
 20 & 21 & 22 & 23 \\
 24 & 25 & 26 & 27 \\
 28 & 29 & 30 & 31
 \end{array} \right]
 \end{matrix}$$

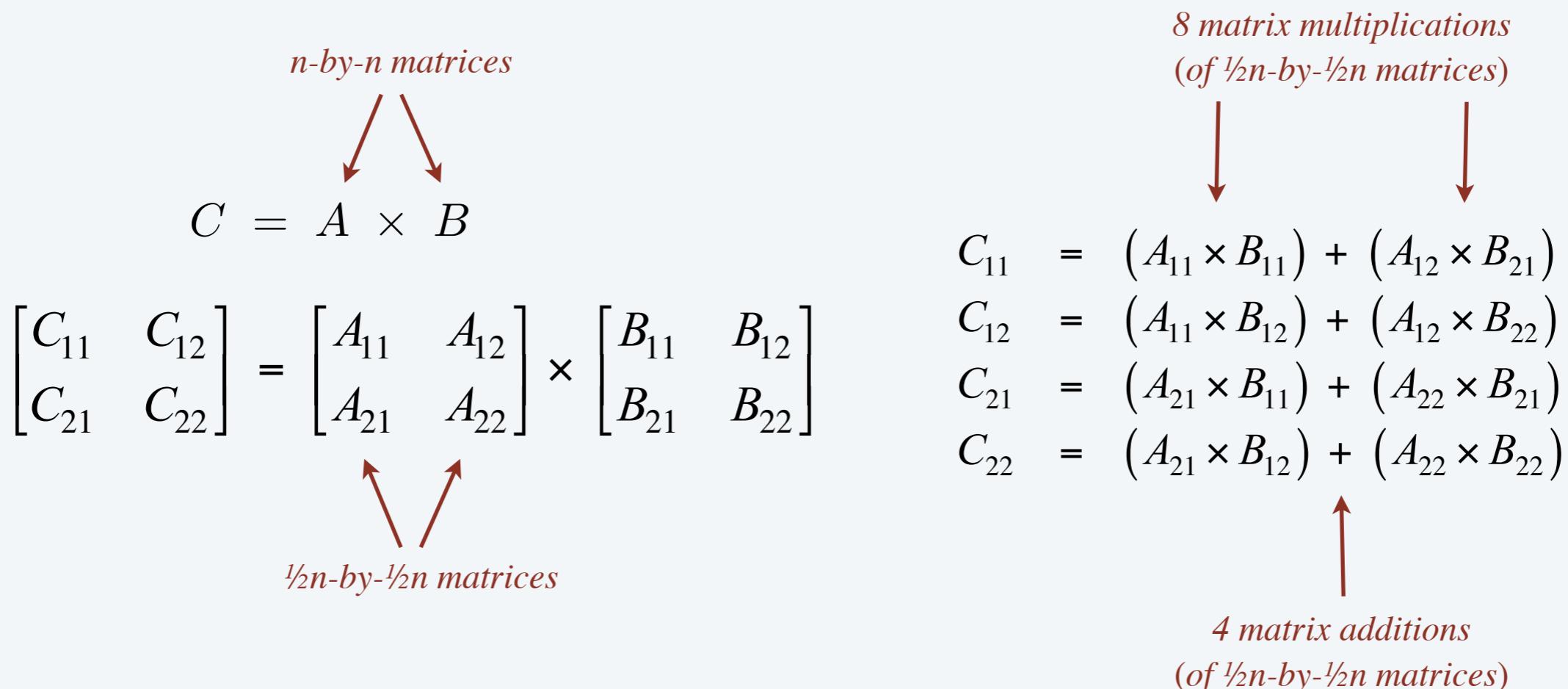
A_{11} A_{12} B_{11}
 B_{21}

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

Block matrix multiplication: warmup

To multiply two n -by- n matrices A and B :

- Divide: partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks.
- Conquer: multiply 8 pairs of $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.



Running time. Apply Case 3 of the master theorem.

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

Strassen's trick

Key idea. Can multiply two 2-by-2 matrices via 7 scalar multiplications (plus 11 additions and 7 subtractions).

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

scalars



$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_1 + P_5 - P_3 - P_7$$

$$P_1 \leftarrow A_{11} \times (B_{12} - B_{22})$$

$$P_2 \leftarrow (A_{11} + A_{12}) \times B_{22}$$

$$P_3 \leftarrow (A_{21} + A_{22}) \times B_{11}$$

$$P_4 \leftarrow A_{22} \times (B_{21} - B_{11})$$

$$P_5 \leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22})$$

$$P_6 \leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$P_7 \leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12})$$



7 scalar multiplications

Pf. $C_{12} = P_1 + P_2$

$$= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22}$$

$$= A_{11} \times B_{12} + A_{12} \times B_{22}. \quad \checkmark$$

Strassen's trick

n-by-*n* $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrix
Key idea. Can multiply two ~~2-by-2~~ matrices via 7 ~~scalar~~ multiplications
 (plus 11 additions and 7 subtractions).

$\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$


$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_1 + P_5 - P_3 - P_7$$

$$P_1 \leftarrow A_{11} \times (B_{12} - B_{22})$$

$$P_2 \leftarrow (A_{11} + A_{12}) \times B_{22}$$

$$P_3 \leftarrow (A_{21} + A_{22}) \times B_{11}$$

$$P_4 \leftarrow A_{22} \times (B_{21} - B_{11})$$

$$P_5 \leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22})$$

$$P_6 \leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$P_7 \leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12})$$



*7 matrix multiplications
 (of $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices)*

Pf. $C_{12} = P_1 + P_2$
 $= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22}$
 $= A_{11} \times B_{12} + A_{12} \times B_{22}. \quad \checkmark$

Strassen's algorithm

assume n is a power of 2
STRASSEN(n, A, B)

IF ($n = 1$) RETURN $A \times B$.

Partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks.

$P_1 \leftarrow \text{STRASSEN}(n / 2, A_{11}, (B_{12} - B_{22}))$.

$P_2 \leftarrow \text{STRASSEN}(n / 2, (A_{11} + A_{12}), B_{22})$.

$P_3 \leftarrow \text{STRASSEN}(n / 2, (A_{21} + A_{22}), B_{11})$.

$P_4 \leftarrow \text{STRASSEN}(n / 2, A_{22}, (B_{21} - B_{11}))$.

$P_5 \leftarrow \text{STRASSEN}(n / 2, (A_{11} + A_{22}), (B_{11} + B_{22}))$.

$P_6 \leftarrow \text{STRASSEN}(n / 2, (A_{12} - A_{22}), (B_{21} + B_{22}))$.

$P_7 \leftarrow \text{STRASSEN}(n / 2, (A_{11} - A_{21}), (B_{11} + B_{12}))$.

$C_{11} = P_5 + P_4 - P_2 + P_6$.

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

$C_{22} = P_1 + P_5 - P_3 - P_7$.

RETURN C .

$\leftarrow 7 T(n / 2) + \Theta(n^2)$

$\leftarrow \Theta(n^2)$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Analysis of Strassen's algorithm

Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two n -by- n matrices.

Gaussian Elimination is not Optimal

VOLKER STRASSEN*

Received December 12, 1968

1. Below we will give an algorithm which computes the coefficients of the product of two square matrices A and B of order n from the coefficients of A and B with less than $4.7 \cdot n^{\log 7}$ arithmetical operations (all logarithms in this paper are for base 2, thus $\log 7 \approx 2.8$; the usual method requires approximately $2n^3$ arithmetical operations). The algorithm induces algorithms for inverting a matrix of order n , solving a system of n linear equations in n unknowns, computing a determinant of order n etc. all requiring less than $\text{const } n^{\log 7}$ arithmetical operations.



Analysis of Strassen's algorithm

Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two n -by- n matrices.

Pf.

- When n is a power of 2, apply Case 1 of the master theorem:

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

- When n is not a power of 2, pad matrices with zeros to be n' -by- n' , where $n \leq n' < 2n$ and n' is a power of 2.

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 10 & 11 & 12 & 0 \\ 13 & 14 & 15 & 0 \\ 16 & 17 & 18 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 84 & 90 & 96 & 0 \\ 201 & 216 & 231 & 0 \\ 318 & 342 & 366 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Strassen's algorithm: practice

Implementation issues.

- Sparsity.
- Caching.
- n not a power of 2.
- Numerical stability.
- Non-square matrices.
- Storage for intermediate submatrices.
- Crossover to classical algorithm when n is “small.”
- Parallelism for multi-core and many-core architectures.

Common misconception. “*Strassen’s algorithm is only a theoretical curiosity.*”

- Apple reports 8x speedup when $n \approx 2,048$.
- Range of instances where it’s useful is a subject of controversy.

Strassen’s Algorithm Reloaded

Jianyu Huang*, Tyler M. Smith*†, Greg M. Henry‡, Robert A. van de Geijn*†

*Department of Computer Science and †Institute for Computational Engineering and Sciences,

The University of Texas at Austin, Austin, TX 78712

Email: jianyu,tms,rvdg@cs.utexas.edu

‡Intel Corporation, Hillsboro, OR 97124

Email: greg.henry@intel.com



Suppose that you could multiply two 3-by-3 matrices with 21 scalar multiplications. How fast could you multiply two n-by-n matrices?

- A. $\Theta(n^{\log_3 21})$
- B. $\Theta(n^{\log_2 21})$
- C. $\Theta(n^{\log_9 21})$
- D. $\Theta(n^2)$



Is it possible to multiply two 3-by-3 matrices using only 21 scalar multiplications?

- A.** Yes.
- B.** No.
- C.** Unknown.

Fast matrix multiplication: theory

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?

A. Yes! [Strassen 1969] $\Theta(n^{\log_2 7}) = O(n^{2.81})$

Q. Multiply two 2-by-2 matrices with 6 scalar multiplications?

A. Impossible. [Hopcroft–Kerr, Winograd 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$

Begun, the decimal wars have. [Pan 1978, Bini et al., Schönhage, ...]

- Two 70-by-70 matrices with 143,640 scalar multiplications. $O(n^{2.7962})$
- Two 48-by-48 matrices with 47,217 scalar multiplications. $O(n^{2.7801})$
- A year later. $O(n^{2.7799})$
- December 1979. $O(n^{2.521813})$
- January 1980. $O(n^{2.521801})$

History of arithmetic complexity of matrix multiplication

year	algorithm	arithmetic operations
1858	“grade school”	$O(n^3)$
1969	Strassen	$O(n^{2.808})$
1978	Pan	$O(n^{2.796})$
1979	Bini	$O(n^{2.780})$
1981	Schönhage	$O(n^{2.522})$
1982	Romani	$O(n^{2.517})$
1982	Coppersmith-Winograd	$O(n^{2.496})$
1986	Strassen	$O(n^{2.479})$
1989	Coppersmith-Winograd	$O(n^{2.3755})$
2010	Strother	$O(n^{2.3737})$
2011	Williams	$O(n^{2.372873})$
2014	Le Gall	$O(n^{2.372864})$
	???	$O(n^{2+\varepsilon})$

galactic algorithms

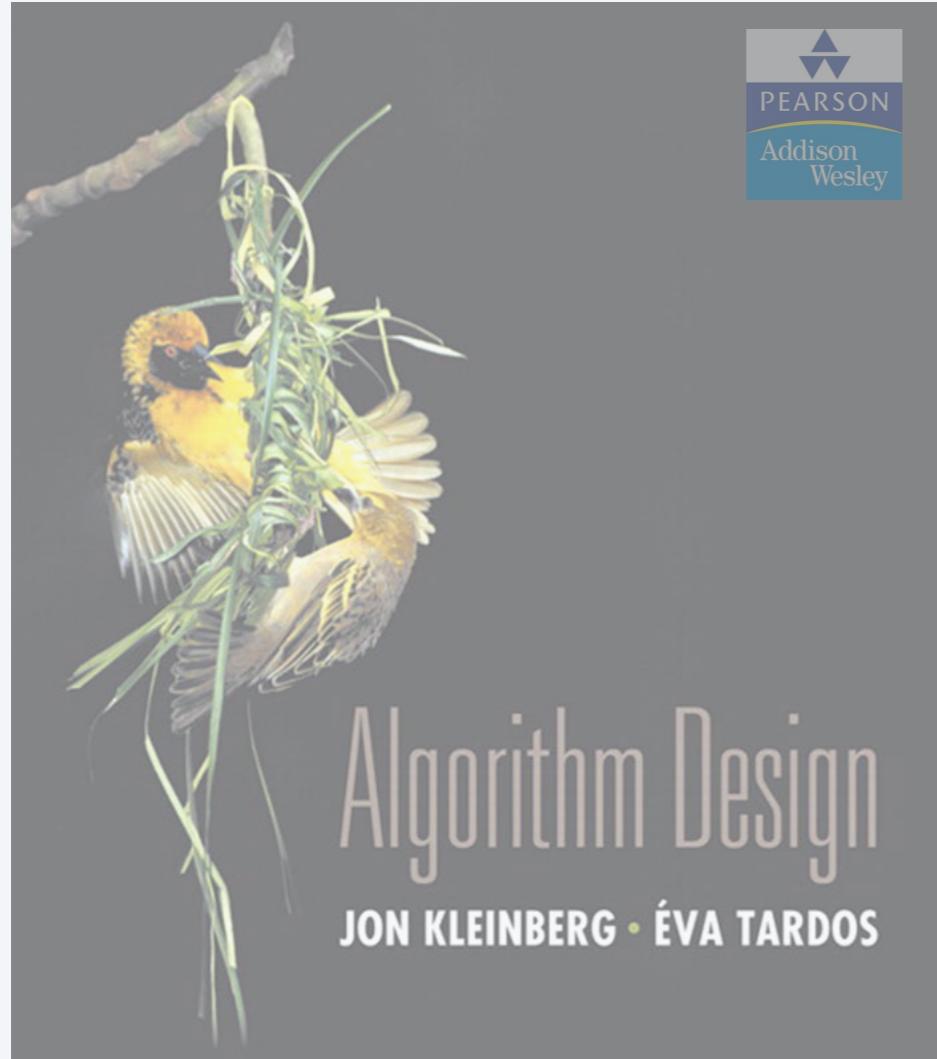
number of arithmetic operations to multiply two n-by-n matrices

Numeric linear algebra reductions

Matrix multiplication. Given two n -by- n matrices, compute their product.

linear algebra problem	expression	arithmetic complexity
matrix multiplication	$A \times B$	$MM(n)$
matrix squaring	A^2	$\Theta(MM(n))$
matrix inversion	A^{-1}	$\Theta(MM(n))$
determinant	$ A $	$\Theta(MM(n))$
rank	$rank(A)$	$\Theta(MM(n))$
system of linear equations	$Ax = b$	$\Theta(MM(n))$
LU decomposition	$A = L U$	$\Theta(MM(n))$
least squares	$\min \ Ax - b\ _2$	$\Theta(MM(n))$

numerical linear algebra problems with the same
arithmetic complexity $MM(n)$ as matrix multiplication



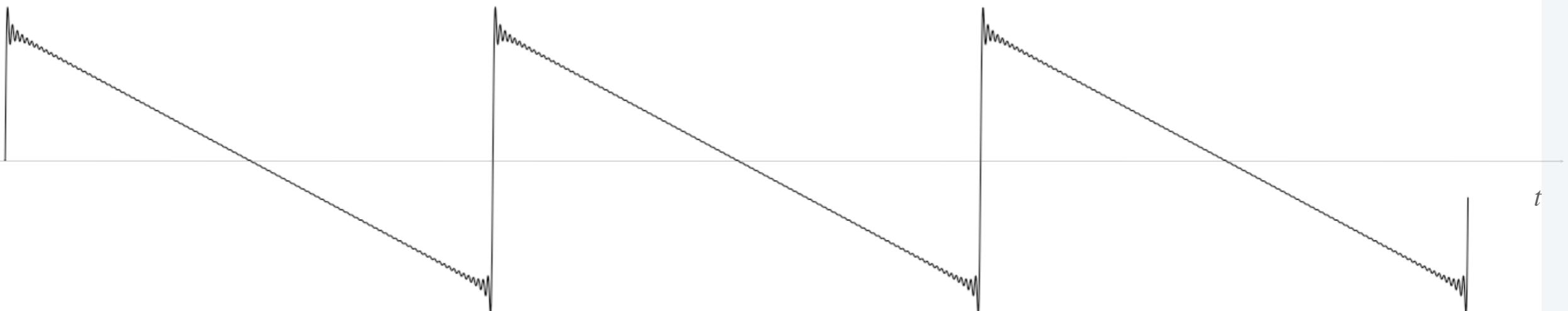
SECTION 5.6

DIVIDE AND CONQUER II

- ▶ *master theorem*
- ▶ *integer multiplication*
- ▶ *matrix multiplication*
- ▶ *convolution and FFT*

Fourier analysis

Fourier theorem. [Fourier, Dirichlet, Riemann] Any (sufficiently smooth) periodic function can be expressed as the sum of a series of sinusoids.



$$y(t) = \frac{2}{\pi} \sum_{k=1}^n \frac{\sin kt}{k} \quad n = 100$$

Euler's identity

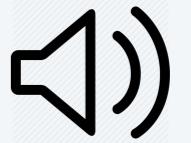
Euler's identity. $e^{ix} = \cos x + i \sin x.$

Sinusoids. Sum of sine and cosines = sum of complex exponentials.

Time domain vs. frequency domain

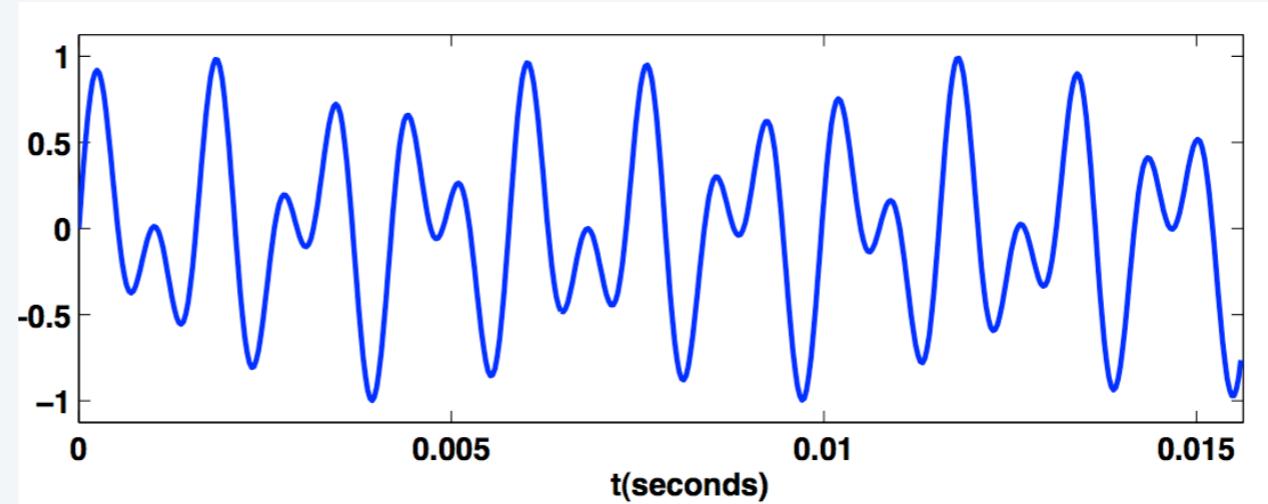
Signal. [touch tone button 1]

$$y(t) = \frac{1}{2} \sin(2\pi \cdot 697 t) + \frac{1}{2} \sin(2\pi \cdot 1209 t)$$



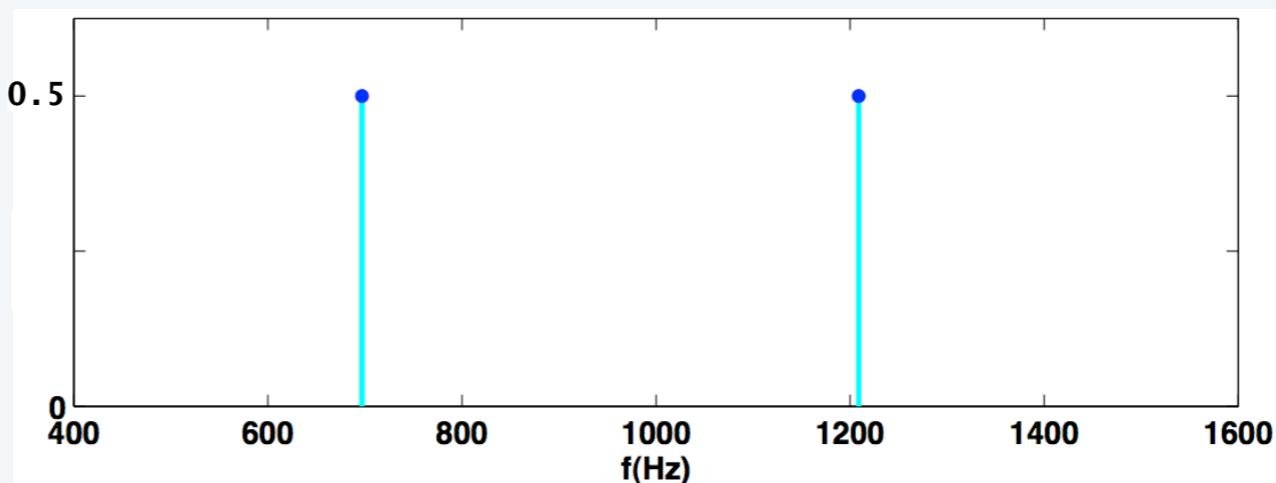
Time domain.

sound
pressure



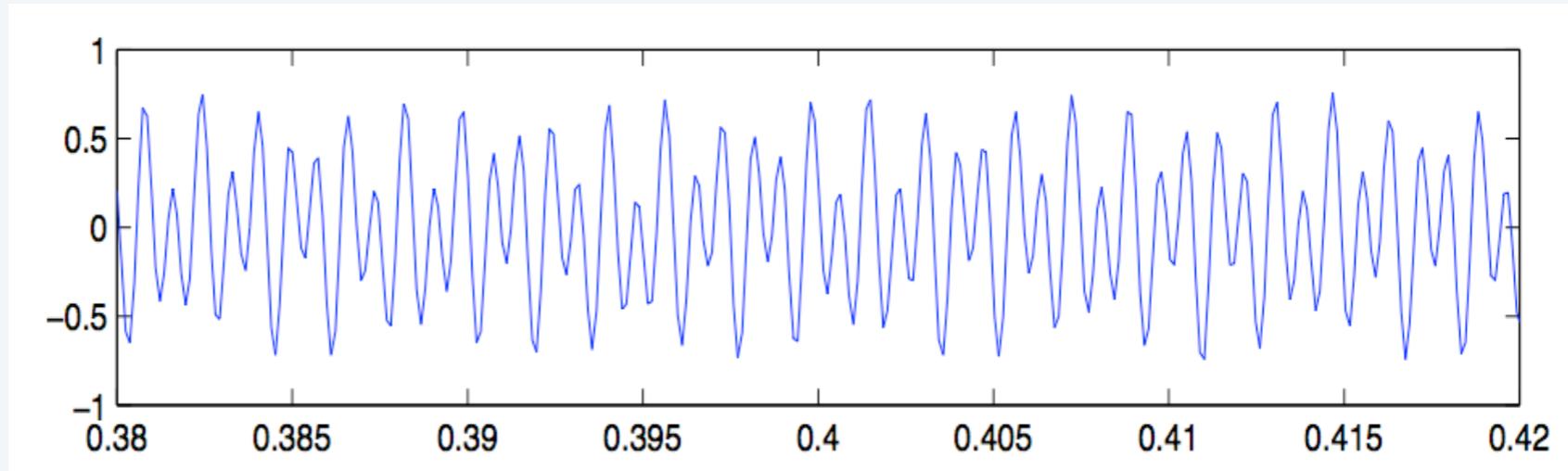
Frequency domain.

amplitude

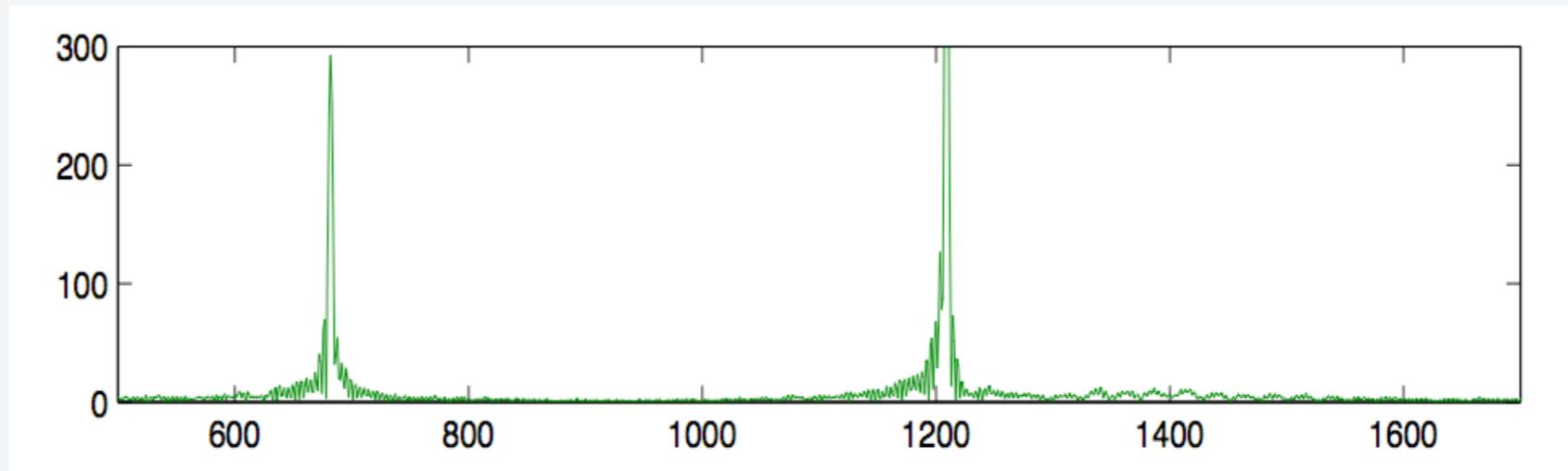


Time domain vs. frequency domain

Signal. [recording, 8192 samples per second]



Magnitude of discrete Fourier transform.



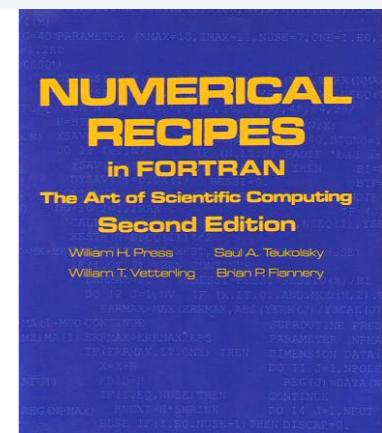
Fast Fourier transform

FFT. Fast way to convert between time domain and frequency domain.

Alternate viewpoint. Fast way to multiply and evaluate polynomials.

we take this approach

“If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it.” — Numerical Recipes



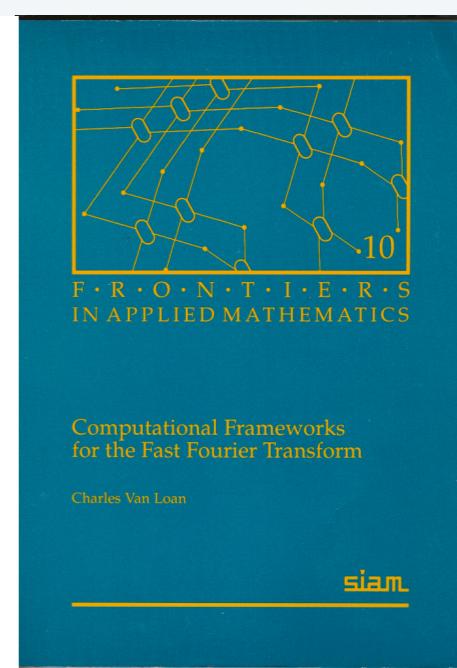
Fast Fourier transform: applications

Applications.

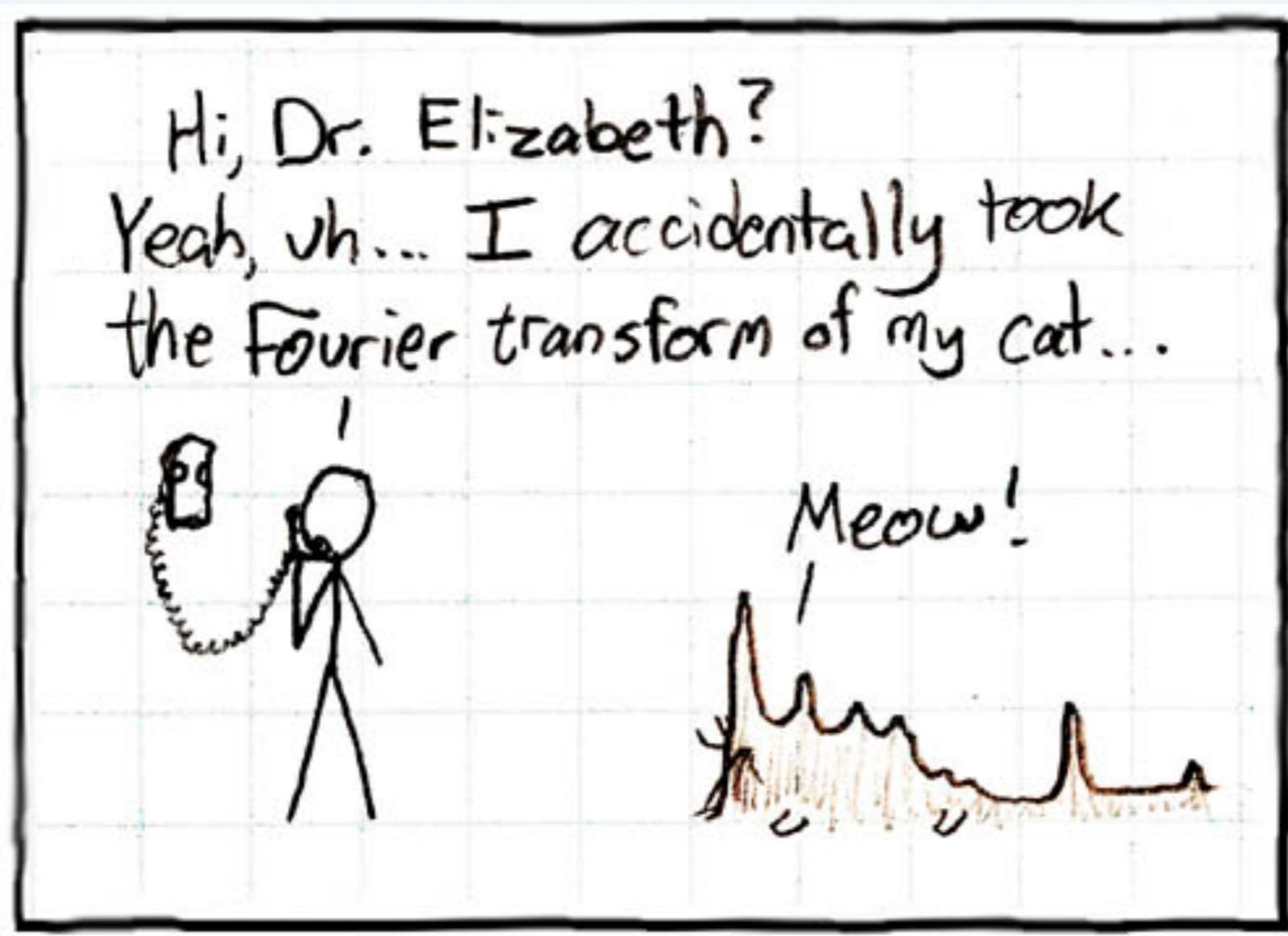
- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry, ...
- Digital media. [DVD, JPEG, MP3, H.264]
- Medical diagnostics. [MRI, CT, PET scans, ultrasound]
- Numerical solutions to Poisson's equation.
- Integer and polynomial multiplication.
- Shor's quantum factoring algorithm.
- ...

“The FFT is one of the truly great computational developments of [the 20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT.”

— Charles van Loan



Fast Fourier transform: applications



<https://xkcd.com/26>

Fast Fourier transform: brief history

Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.

Runge–König (1924). Laid theoretical groundwork.

Danielson–Lanczos (1942). Efficient algorithm, x-ray crystallography.

Cooley–Tukey (1965). Detect nuclear tests in Soviet Union and track submarines. Rediscovered and popularized FFT.



An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N -vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2 .



Importance not fully realized until emergence of digital computers.

Polynomials: coefficient representation

Univariate polynomial. [coefficient representation]

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_{n-1}x^{n-1}$$

Addition. $O(n)$ arithmetic operations.

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \dots + (a_{n-1} + b_{n-1})x^{n-1}$$

Evaluation. $O(n)$ using Horner's method.

$$A(x) = a_0 + (x(a_1 + x(a_2 + \dots + x(a_{n-2} + x(a_{n-1})) \dots)))$$

```
double val = 0.0;
for (int j = n-1; j >= 0; j--)
    val = a[j] + (x * val);
```

Multiplication (linear convolution). $O(n^2)$ using brute force.

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i \text{ where } c_i = \sum_{j=0}^i a_j b_{i-j}$$



What was the subject of Gauss' Ph.D thesis?

- A. Gaussian elimination.
- B. Fast Fourier transform.
- C. Prime number theorem.
- D. Cauchy integral theorem.
- E. Fundamental theorem of algebra.
- F. Angle-preserving maps.
- G. Method of least squares.
- H. Non-Euclidean geometry.
- I. Constructing a regular heptadecagon with straightedge and compass.

A modest Ph.D. dissertation title

DEMONSTRATIO NOVA
THEOREMATIC
OMNEM FVNCTIONEM ALGEBRAICAM
RATIONALEM INTEGRAM
VNIVS VARIABILIS
IN FACTORES REALES PRIMI VEL SECUNDI GRADVS
RESOLVI POSSE

AVCTORE
CAROLO FRIDERICO GAVSS
HELMSTADII
APVD C. G. FLECKEISEN. 1799

1.

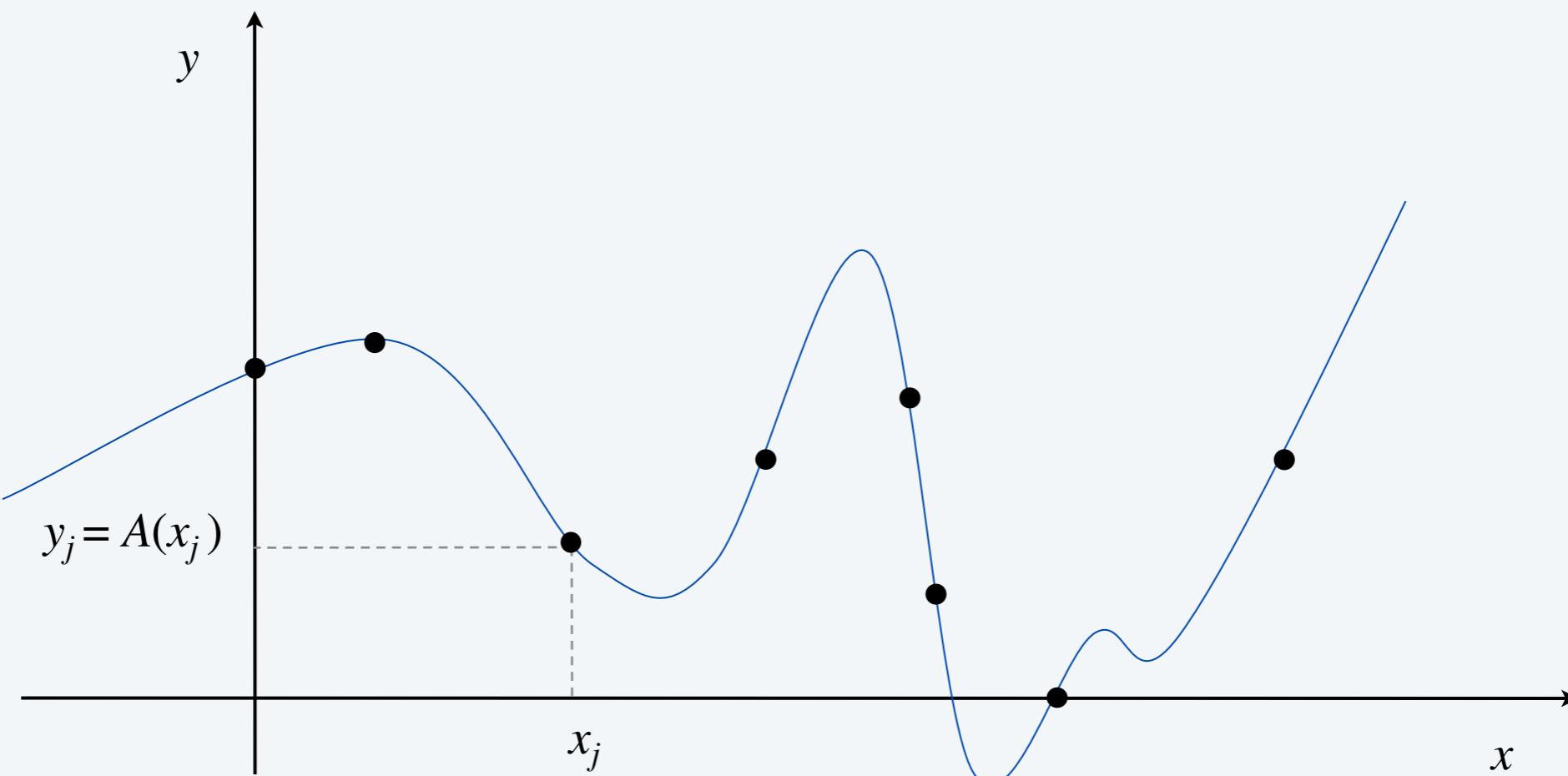
Quaelibet aequatio algebraica determinata reduci potest ad formam $x^m + Ax^{m-1} + Bx^{m-2} + \dots + M = 0$, ita vt m sit numerus integer positivus. Si partem primam huius aequationis per X denotamus, aequationique $X=0$ per plures valores inaequales ipsius x satisfieri supponimus, puta ponendo $x=\alpha, x=\beta, x=\gamma$ etc. functio X per productum e factoribus $x-\alpha, x-\beta, x-\gamma$ etc. diuisibilis erit. Vice versa, si productum e pluribus factoribus simplicibus $x-\alpha, x-\beta, x-\gamma$ etc. functionem X metitur: aequationi $X=0$ satisfiet, aquando ipsam x cuicunque quantitatum α, β, γ etc. Denique si X producto ex m factoribus talibus simplicibus aequalis est (siue omnes diuersi sint, siue quidam ex ipsis identici): alii factores simplices praeter hos functionem X metiri non poterunt. Quamobrem aequatio m^{ti} gradus plures quam m radices habere nequit; simul vero patet, aequationem m^{ti} gradus pauciores radices habere posse, etsi X in m factores simplices resolubilis sit:

“ New proof of the theorem that every algebraic rational integral function in one variable can be resolved into real factors of the first or the second degree. ”

Polynomials: point-value representation

Fundamental theorem of algebra. A degree n univariate polynomial with complex coefficients has exactly n complex roots.

Corollary. A degree $n - 1$ univariate polynomial $A(x)$ is uniquely specified by its evaluation at n distinct values of x .



Polynomials: point-value representation

Univariate polynomial. [point-value representation]

$A(x)$: $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$

$B(x)$: $(x_0, z_0), \dots, (x_{n-1}, z_{n-1})$

Addition. $O(n)$ arithmetic operations.

$A(x) + B(x)$: $(x_0, y_0 + z_0), \dots, (x_{n-1}, y_{n-1} + z_{n-1})$

Multiplication. $O(n)$, but represent $A(x)$ and $B(x)$ using $2n$ points.

$A(x) \times B(x)$: $(x_0, y_0 \times z_0), \dots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})$

Evaluation. $O(n^2)$ using Lagrange's formula.

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

← not used

Converting between two representations

Tradeoff. Either fast evaluation or fast multiplication. We want both!

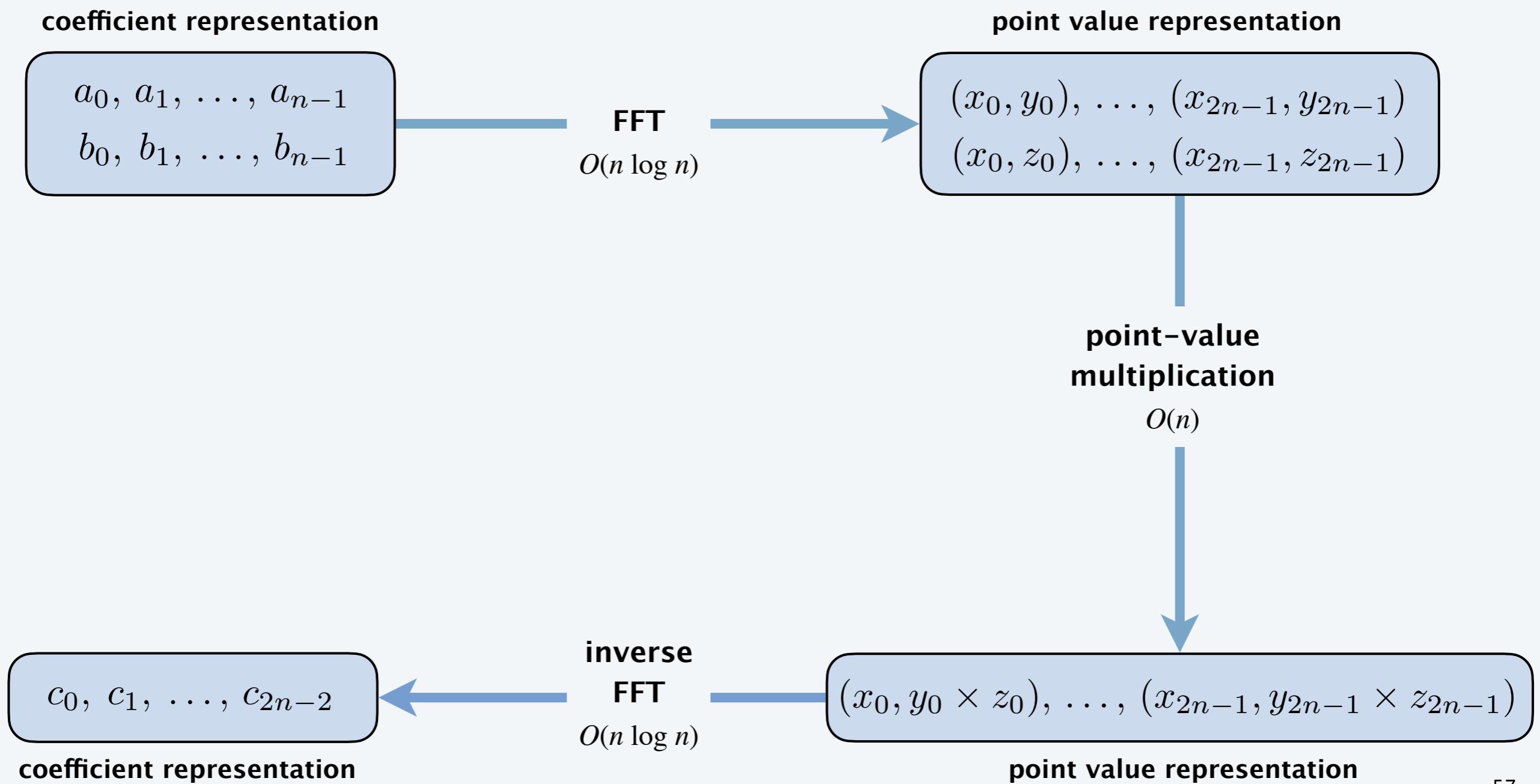
representation	multiply	evaluate
coefficient	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n^2)$

Goal. Efficient conversion between two representations \Rightarrow all ops fast.



Converting between two representations

Application. Polynomial multiplication (coefficient representation).



Converting between two representations: brute force

Coefficient \Rightarrow point-value. Given a polynomial $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, x_{n-1} .

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Running time. $O(n^2)$ via matrix–vector multiply (or n Horner’s).

Converting between two representations: brute force

Point-value \Rightarrow coefficient. Given n distinct points x_0, \dots, x_{n-1} and values y_0, \dots, y_{n-1} , find unique polynomial $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, that has given values at given points.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$



Vandermonde matrix is invertible iff x_i distinct

Running time. $O(n^3)$ via Gaussian elimination.



or $O(n^{2.38})$ via fast matrix multiplication



Which divide-and-conquer approach to use to multiply polynomials?

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$$

- A. Divide polynomial into low- and high-degree terms.

$$A_{low}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

$$A_{high}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3.$$

- B. Divide polynomial into even- and odd-degree terms.

$$A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$$

$$A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$$

- C. Either A or B.

- D. Neither A nor B.

Divide-and-conquer

Decimation in time. Divide into even- and odd-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2).$

Cooley-Tukey radix 2 FFT

Decimation in frequency. Divide into low- and high-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{low}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$
- $A_{high}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3.$
- $A(x) = A_{low}(x) + x^4 A_{high}(x).$

Sande-Tukey radix 2 FFT

Coefficient to point-value representation: intuition

Coefficient \Rightarrow point-value. Given a polynomial $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, x_{n-1} . ← we get to choose which ones!

Divide. Break up polynomial into even- and odd-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2).$
- $A(-x) = A_{even}(x^2) - x A_{odd}(x^2).$

Intuition. Choose two points to be ± 1 .

- $A(1) = A_{even}(1) + 1 A_{odd}(1).$
 - $A(-1) = A_{even}(1) - 1 A_{odd}(1).$
- ← Can evaluate polynomial of degree $n-1$ at 2 points by evaluating two polynomials of degree $\frac{1}{2}n - 1$ at only 1 point.

Coefficient to point-value representation: intuition

Coefficient \Rightarrow point-value. Given a polynomial $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, x_{n-1} . ← we get to choose which ones!

Divide. Break up polynomial into even- and odd-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2).$
- $A(-x) = A_{even}(x^2) - x A_{odd}(x^2).$

Intuition. Choose four **complex** points to be $\pm 1, \pm i$.

- $A(1) = A_{even}(1) + 1 A_{odd}(1).$
- $A(-1) = A_{even}(1) - 1 A_{odd}(1).$
- $A(i) = A_{even}(-1) + i A_{odd}(-1).$
- $A(-i) = A_{even}(-1) - i A_{odd}(-1).$

← Can evaluate polynomial of degree $n-1$ at 4 points by evaluating two polynomials of degree $\frac{1}{2}n - 1$ at only 2 points.

Discrete Fourier transform

Coefficient \Rightarrow point-value. Given a polynomial $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, x_{n-1} . \leftarrow we get to choose which ones!

Key idea. Choose $x_k = \omega^k$ where ω is principal n^{th} root of unity.

$$y_k = A(\omega^k) \rightarrow \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

\uparrow \uparrow
DFT Fourier matrix F_n

Roots of unity

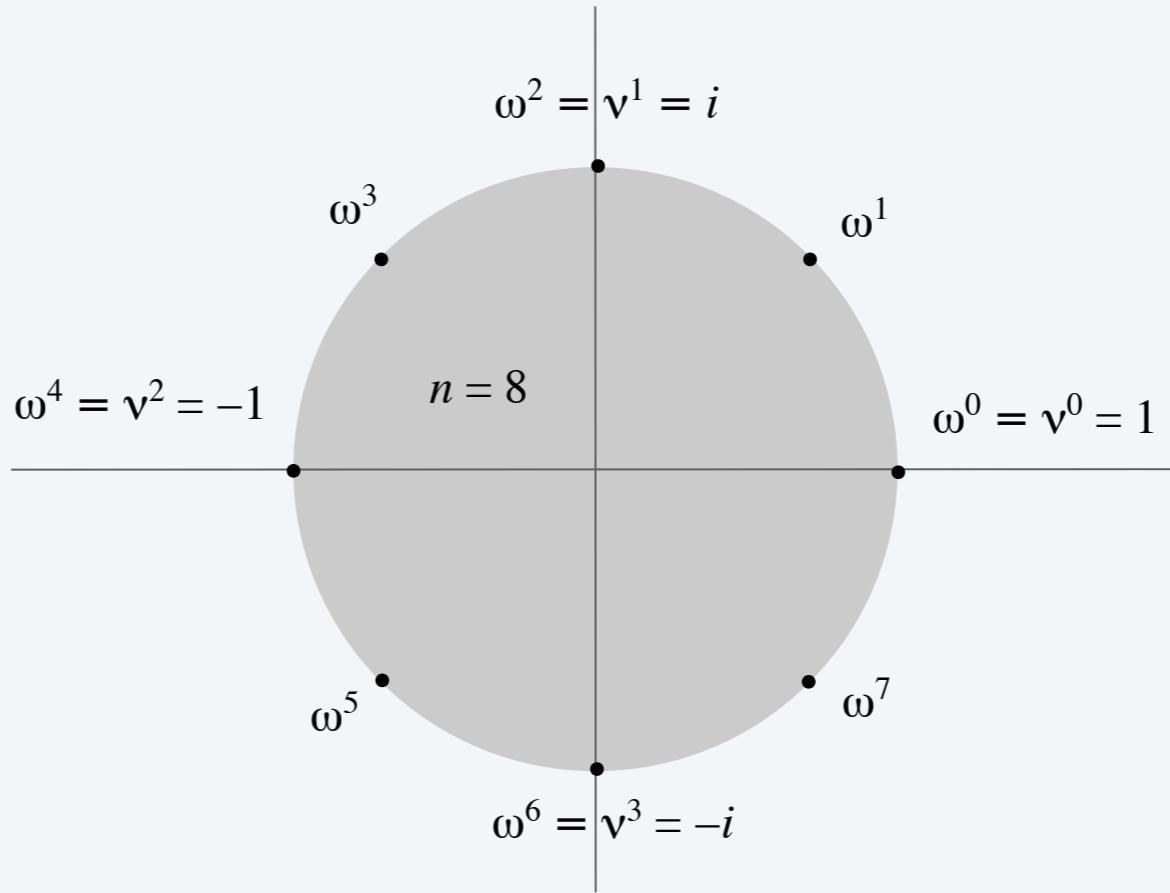
Def. An n^{th} root of unity is a complex number x such that $x^n = 1$.

Fact. The n^{th} roots of unity are: $\omega^0, \omega^1, \dots, \omega^{n-1}$ where $\omega = e^{2\pi i / n}$.

Pf. $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$.

common alternative: $\omega = e^{-2\pi i / n}$

Fact. The $\frac{1}{2}n^{\text{th}}$ roots of unity are: $v^0, v^1, \dots, v^{n/2-1}$ where $v = \omega^2 = e^{4\pi i / n}$.



Fast Fourier transform

Goal. Evaluate a degree $n - 1$ polynomial $A(x) = a_0 + \dots + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \dots, \omega^{n-1}$.

Divide. Break up polynomial into even- and odd-degree terms.

- $A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + \dots + a_{n-2} x^{n/2-1}$.
- $A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + \dots + a_{n-1} x^{n/2-1}$.
- $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2)$.
- $A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2)$.

Conquer. Evaluate $A_{\text{even}}(x)$ and $A_{\text{odd}}(x)$ at the $\frac{1}{2}n^{\text{th}}$ roots of unity: $\nu^0, \nu^1, \dots, \nu^{n/2-1}$.

Combine.

- $$\nu^k = (\omega^k)^2$$
- 
- $y_k = A(\omega^k) = A_{\text{even}}(\nu^k) + \omega^k A_{\text{odd}}(\nu^k), \quad 0 \leq k < n/2$.
 - $y_{k+\frac{1}{2}n} = A(\omega^{k+\frac{1}{2}n}) = A_{\text{even}}(\nu^k) - \omega^k A_{\text{odd}}(\nu^k), \quad 0 \leq k < n/2$.

$$\begin{array}{c} \uparrow \\ A(-\omega^k) \end{array}$$

FFT: implementation

Goal. Evaluate a degree $n - 1$ polynomial $A(x) = a_0 + \dots + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \dots, \omega^{n-1}$.

- $y_k = A(\omega^k) = A_{\text{even}}(\mathbf{v}^k) + \omega^k A_{\text{odd}}(\mathbf{v}^k), \quad 0 \leq k < n/2.$
- $y_{k+\frac{n}{2}} = A(\omega^{k+\frac{n}{2}}) = A_{\text{even}}(\mathbf{v}^k) - \omega^k A_{\text{odd}}(\mathbf{v}^k), \quad 0 \leq k < n/2.$

FFT($n, a_0, a_1, a_2, \dots, a_{n-1}$)

IF ($n = 1$) **RETURN** a_0 .

$(e_0, e_1, \dots, e_{n/2-1}) \leftarrow \text{FFT}(n / 2, a_0, a_2, a_4, \dots, a_{n-2}).$

$(d_0, d_1, \dots, d_{n/2-1}) \leftarrow \text{FFT}(n / 2, a_1, a_3, a_5, \dots, a_{n-1}).$

FOR $k = 0$ **TO** $n / 2 - 1$.

$$\omega^k \leftarrow e^{2\pi i k/n}.$$

$$y_k \leftarrow e_k + \omega^k d_k.$$

$$y_{k+n/2} \leftarrow e_k - \omega^k d_k.$$

RETURN $(y_0, y_1, y_2, \dots, y_{n-1})$.

$2 T(n / 2)$

$\Theta(n)$

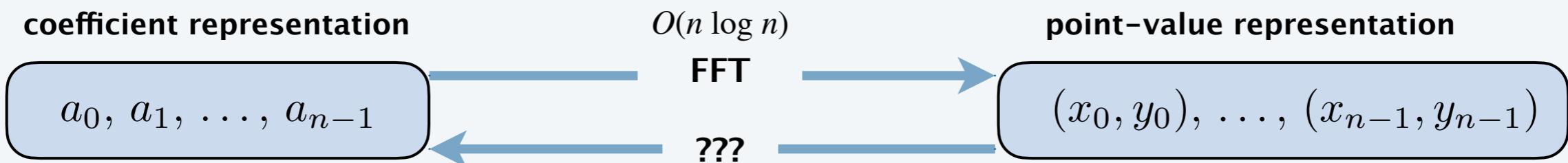
FFT: summary

Theorem. The FFT algorithm evaluates a degree $n - 1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ arithmetic operations and $O(n)$ extra space.

Pf.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

assumes n is a power of 2

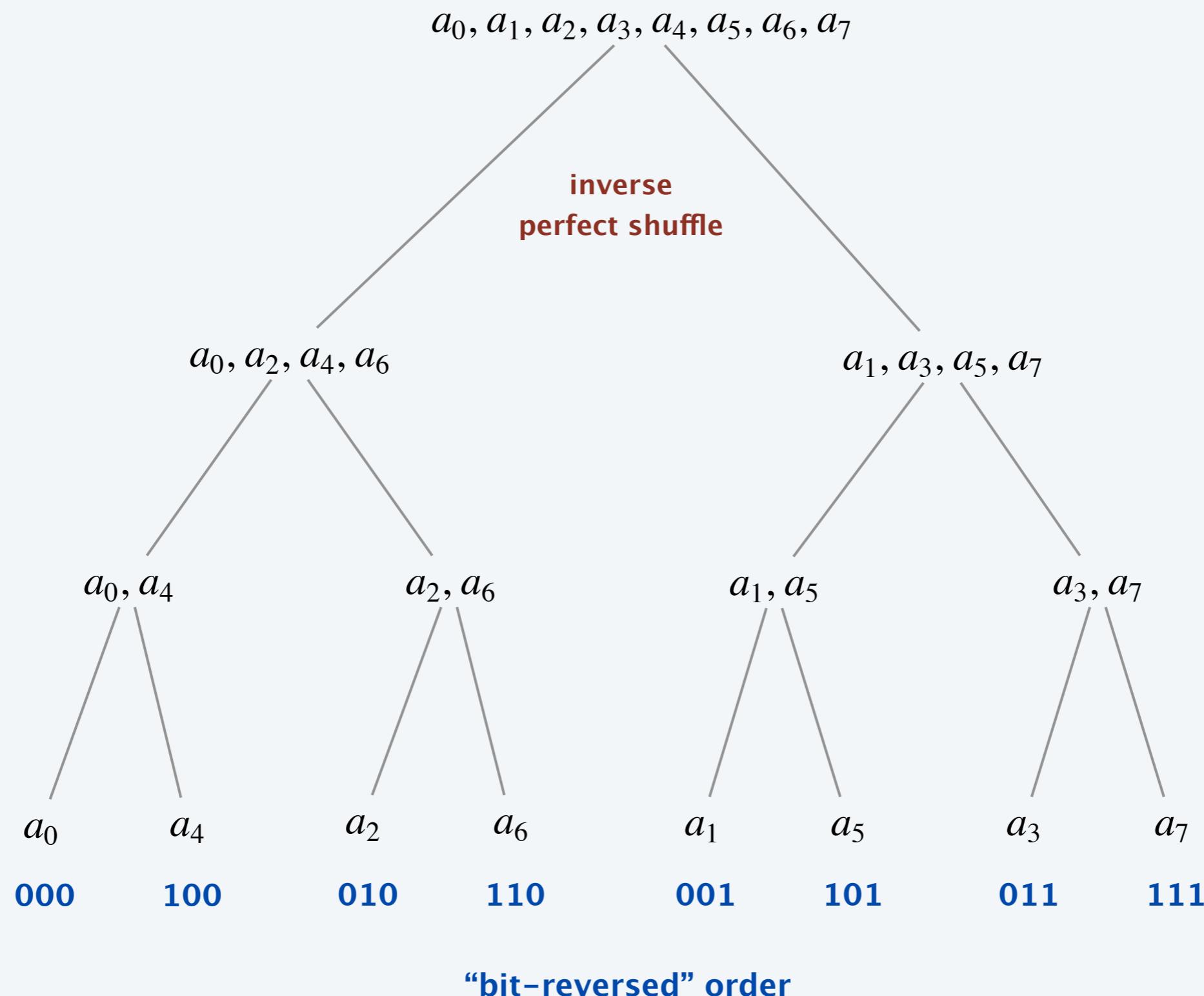




When computing the FFT of $(a_0, a_1, a_2, \dots, a_7)$, which are the first two coefficients involved in an arithmetic operation?

- A. a_0 and a_1 .
- B. a_0 and a_2 .
- C. a_0 and a_4 .
- D. a_0 and a_7 .
- E. None of the above.

FFT: recursion tree



FFT: Fourier matrix decomposition

Alternative viewpoint. FFT is a recursive decomposition of Fourier matrix.

$$F_n = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \quad a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Fourier matrix

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \quad D_n = \begin{bmatrix} \omega^0 & 0 & 0 & \cdots & 0 \\ 0 & \omega^1 & 0 & \cdots & 0 \\ 0 & 0 & \omega^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \omega^{n-1} \end{bmatrix}$$

DFT

$$y = F_n a = \begin{bmatrix} I_{n/2} & D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2} a_{even} \\ F_{n/2} a_{odd} \end{bmatrix}$$

Inverse discrete Fourier transform

Point-value \Rightarrow coefficient. Given n distinct points x_0, \dots, x_{n-1} and values y_0, \dots, y_{n-1} , find unique polynomial $a_0 + a_1x + \dots + a_{n-1}x^{n-1}$, that has given values at given points.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Inverse discrete Fourier transform

Claim. Inverse of Fourier matrix F_n is given by following formula:

$$G_n = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

F_n / \sqrt{n} is a unitary matrix

Consequence. To compute the inverse FFT, apply the same algorithm but use $\omega^{-1} = e^{-2\pi i / n}$ as principal n^{th} root of unity (and divide the result by n).

Inverse FFT: proof of correctness

Claim. F_n and G_n are inverses.

Pf.

$$(F_n G_n)_{kk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj} \omega^{-jk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$$

summation lemma (below)

Summation lemma. Let ω be a principal n^{th} root of unity. Then

$$\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} n & \text{if } k \equiv 0 \pmod{n} \\ 0 & \text{otherwise} \end{cases}$$

Pf.

- If k is a multiple of n , then $\omega^k = 1 \Rightarrow$ series sums to n .
- Each n^{th} root of unity ω^k is a root of $x^n - 1 = (x - 1)(1 + x + x^2 + \dots + x^{n-1})$.
- if $\omega^k \neq 1$, then $1 + \omega^k + \omega^{k(2)} + \dots + \omega^{k(n-1)} = 0 \Rightarrow$ series sums to 0. ▀

Inverse FFT: implementation

Note. Need to divide result by n .

INVERSE-FFT($n, y_0, y_1, y_2, \dots, y_{n-1}$)

IF ($n = 1$) **RETURN** y_0 .

$(e_0, e_1, \dots, e_{n/2-1}) \leftarrow \text{INVERSE-FFT}(n / 2, y_0, y_2, y_4, \dots, y_{n-2})$.

$(d_0, d_1, \dots, d_{n/2-1}) \leftarrow \text{INVERSE-FFT}(n / 2, y_1, y_3, y_5, \dots, y_{n-1})$.

FOR $k = 0$ **TO** $n / 2 - 1$.

$$\omega^k \leftarrow e^{-2\pi i k / n}.$$

$$a_k \leftarrow e_k + \omega^k d_k.$$

$$a_{k+n/2} \leftarrow e_k - \omega^k d_k.$$

RETURN $(a_0, a_1, a_2, \dots, a_{n-1})$.

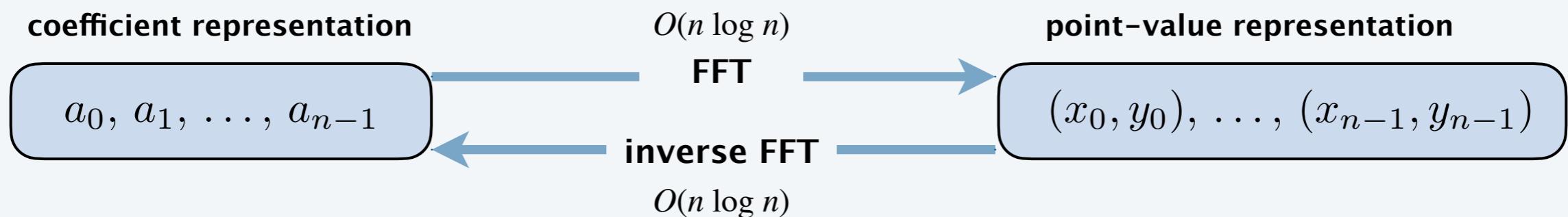
switch roles of a_i and y_i

Inverse FFT: summary

Theorem. The inverse FFT algorithm interpolates a degree $n - 1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ arithmetic operations.

↑
assumes n is a power of 2

Corollary. Can convert between coefficient and point-value representations in $O(n \log n)$ arithmetic operations.

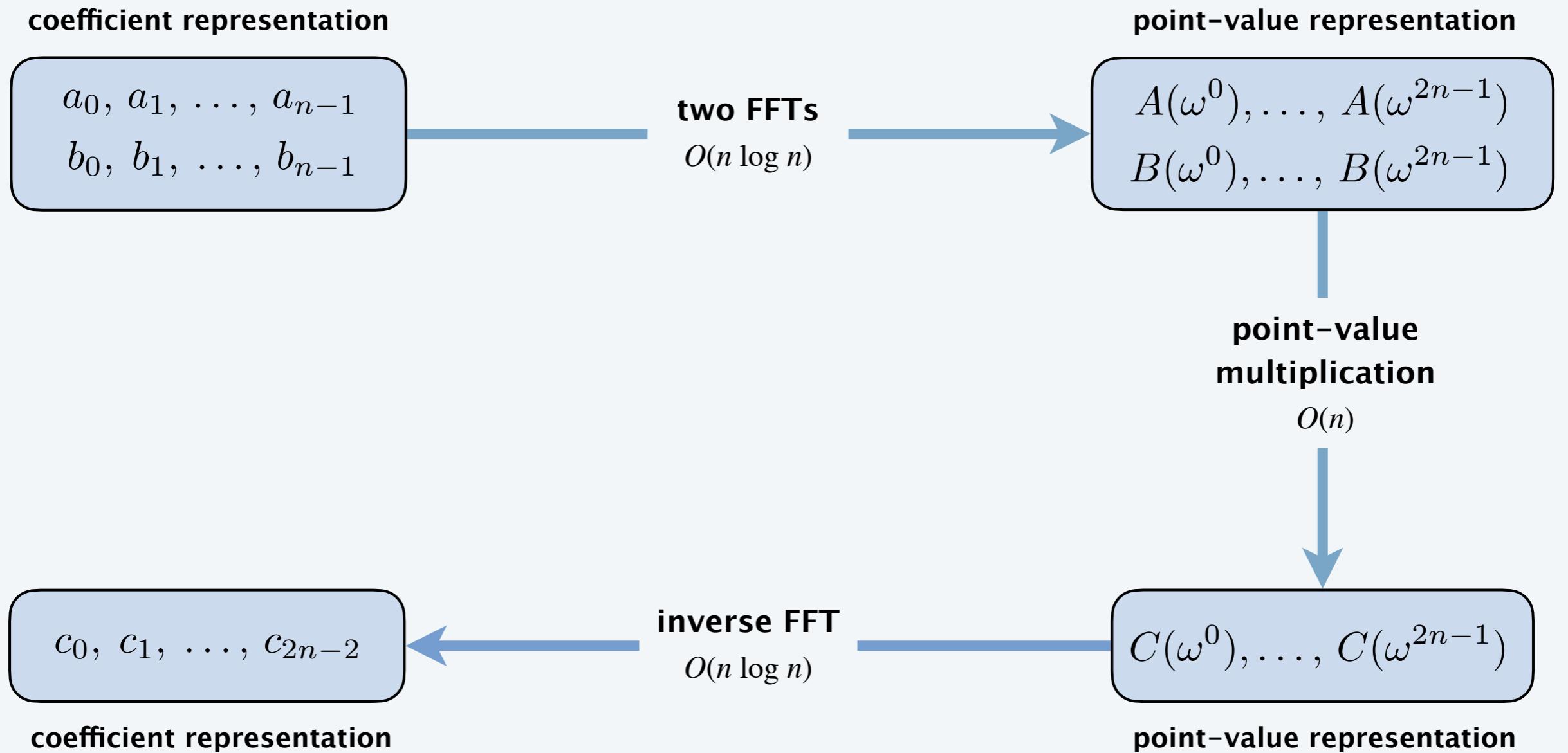


Polynomial multiplication

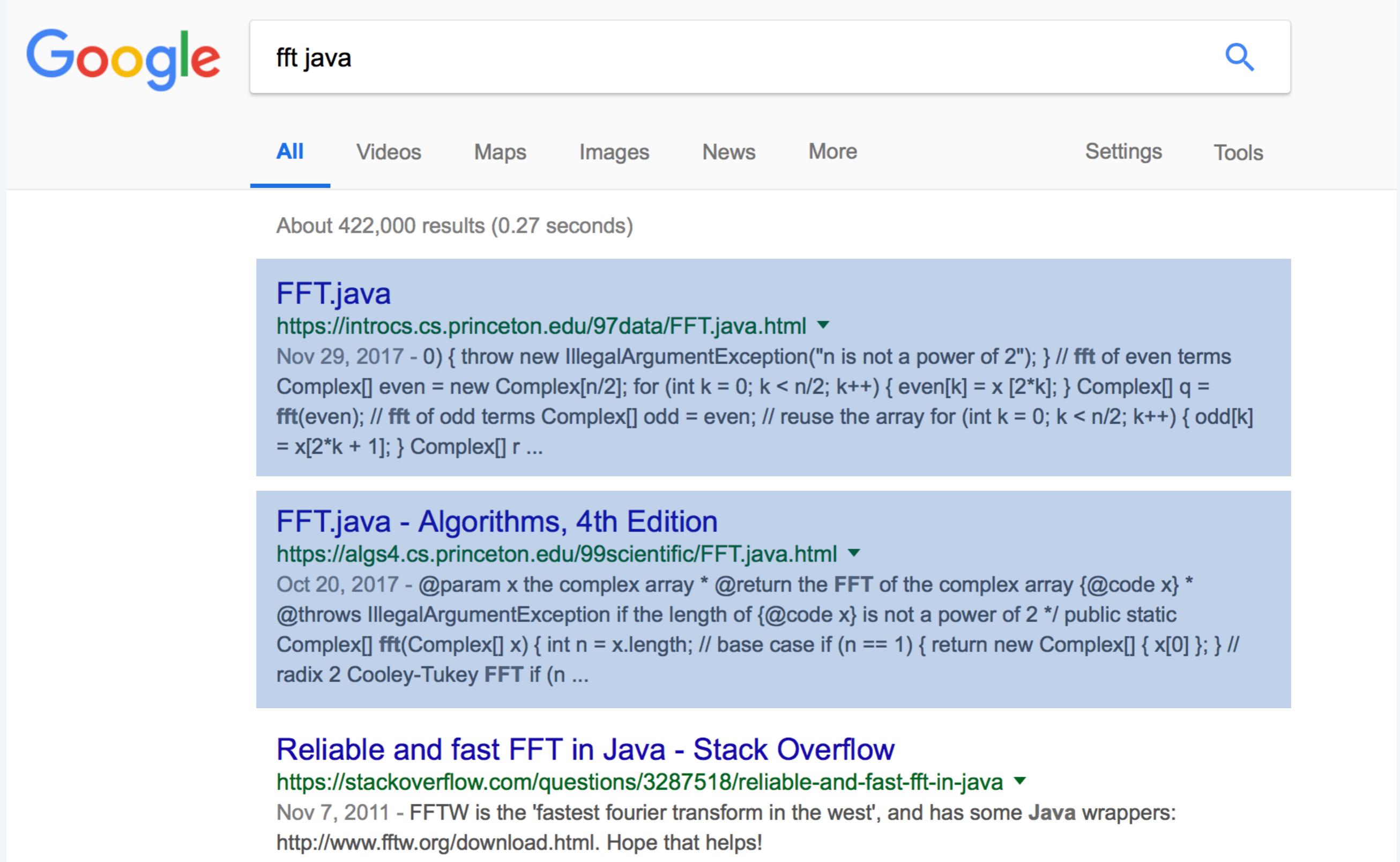
Theorem. Given two polynomials $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$ and $B(x) = b_0 + b_1 x + \dots + b_{n-1} x^{n-1}$ of degree $n - 1$, can multiply them in $O(n \log n)$ arithmetic operations.

pad with 0s to make
 n a power of 2

Pf.



FFT in practice ?



A screenshot of a Google search results page. The search query "fft java" is entered in the search bar. The "All" tab is selected, showing approximately 422,000 results found in 0.27 seconds. The first result is a snippet from "FFT.java" located at <https://introcs.cs.princeton.edu/97data/FFT.java.html>. It shows Java code for performing an FFT on a complex array. The second result is a snippet from "FFT.java - Algorithms, 4th Edition" located at <https://algs4.cs.princeton.edu/99scientific/FFT.java.html>. It shows Java code for the Cooley-Tukey FFT algorithm. The third result is a snippet from "Reliable and fast FFT in Java - Stack Overflow" located at <https://stackoverflow.com/questions/3287518/reliable-and-fast-fft-in-java>. It discusses FFTW and provides a link to its Java wrappers.

Google search results for "fft java":

About 422,000 results (0.27 seconds)

FFT.java
<https://introcs.cs.princeton.edu/97data/FFT.java.html> ▾

Nov 29, 2017 - 0) { throw new IllegalArgumentException("n is not a power of 2"); } // fft of even terms
Complex[] even = new Complex[n/2]; for (int k = 0; k < n/2; k++) { even[k] = x[2*k]; } Complex[] q =
fft(even); // fft of odd terms Complex[] odd = even; // reuse the array for (int k = 0; k < n/2; k++) { odd[k]
= x[2*k + 1]; } Complex[] r ...

FFT.java - Algorithms, 4th Edition
<https://algs4.cs.princeton.edu/99scientific/FFT.java.html> ▾

Oct 20, 2017 - @param x the complex array * @return the FFT of the complex array {@code x} *
@throws IllegalArgumentException if the length of {@code x} is not a power of 2 */ public static
Complex[] fft(Complex[] x) { int n = x.length; // base case if (n == 1) { return new Complex[] { x[0] }; } //
radix 2 Cooley-Tukey FFT if (n ...

Reliable and fast FFT in Java - Stack Overflow
<https://stackoverflow.com/questions/3287518/reliable-and-fast-fft-in-java> ▾

Nov 7, 2011 - FFTW is the 'fastest fourier transform in the west', and has some Java wrappers:
<http://www.fftw.org/download.html>. Hope that helps!

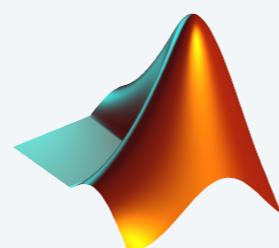
FFT in practice

Fastest Fourier transform in the West. [Frigo–Johnson]

- Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won 1999 Wilkinson Prize for Numerical Software.
- Portable, competitive with vendor-tuned code.

Implementation details.

- Core algorithm is an in-place, nonrecursive version of Cooley–Tukey.
- Instead of executing a fixed algorithm, it evaluates the hardware and uses a special-purpose compiler to generate an optimized algorithm catered to “shape” of the problem.
- Runs in $O(n \log n)$ time, even when n is prime.
- Multidimensional FFTs.
- Parallelism.



<http://www.fftw.org>

Top 10 algorithms of the 20th century

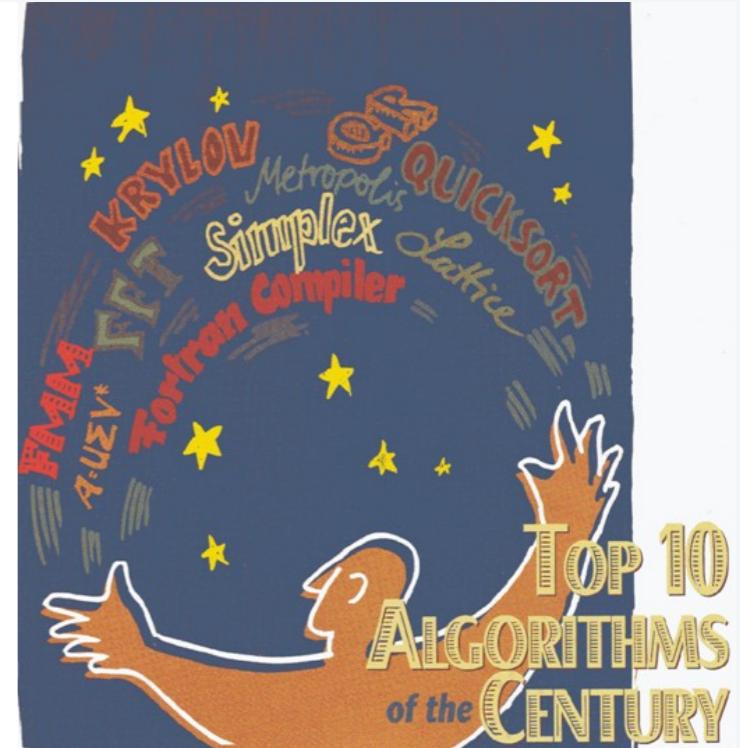
FROM THE
EDITORS

THE JOY OF ALGORITHMS

Francis Sullivan, Associate Editor-in-Chief



THE THEME OF THIS FIRST-OF-THE-CENTURY ISSUE OF *COMPUTING IN SCIENCE & ENGINEERING* IS ALGORITHMS. IN FACT, WE WERE BOLD ENOUGH—AND PERHAPS FOOLISH ENOUGH—TO CALL THE 10 EXAMPLES WE'VE SELECTED “THE TOP 10 ALGORITHMS OF THE CENTURY.”



Daniel Rockmore describes the FFT as an algorithm “the whole family can use.” The FFT is perhaps the most ubiquitous algorithm in use today to analyze and manipulate digital or discrete data. The FFT takes the operation count for discrete Fourier transform from $O(N^2)$ to $O(N \log N)$.

Integer multiplication, redux

Integer multiplication. Given two n -bit integers $a = a_{n-1} \dots a_1 a_0$ and $b = b_{n-1} \dots b_1 b_0$, compute their product $a \cdot b$.

Convolution algorithm.

- Form two polynomials.
 - Note: $a = A(2)$, $b = B(2)$.
 - Compute $C(x) = A(x) \cdot B(x)$.
 - Evaluate $C(2) = a \cdot b$.
 - Running time: $O(n \log n)$ floating-point operations.
- $$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$
- $$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_{n-1}x^{n-1}$$

Theory. [Schönhage–Strassen 1971]

- $O(n \log^2 n)$ bit operations. \leftarrow FFT over complex numbers; need $O(\log n)$ bits of precision
- $O(n \log n \cdot \log \log n)$ bit operations. \leftarrow FFT over ring of integers (modulo a Fermat number)

Practice. [GNU Multiple Precision Arithmetic Library]

Switches to FFT-based algorithm when n is large ($\geq 5\text{--}10K$).

3-SUM (REVISITED)



3-SUM. Given three sets X , Y , and Z of n integers each, determine whether there is a triple $i \in X, j \in Y, k \in Z$ such that $i + j = k$.

Assumption. All integers are between 0 and m .

Goal. $O(m \log m + n \log n)$ time.

$$m = 19, n = 3$$

$$X = \{ 4, 7, 10 \}$$

$$Y = \{ 5, 8, 15 \}$$

$$Z = \{ 4, 13, 19 \}$$

a yes instance

$$(4 + 15 = 19)$$

3-SUM (REVISITED)



An $O(m \log m + n)$ solution.

- Form polynomial $A(x) = a_0 + a_1 x + \dots + a_m x^m$ with $a_i = 1$ iff $i \in X$.
- Form polynomial $B(x) = b_0 + b_1 x + \dots + b_m x^m$ with $b_j = 1$ iff $j \in Y$.
- Compute product/convolution $C(x) = A(x) \times B(x)$.
- The coefficient $c_k =$ number of ways to choose an integer $i \in X$ and an integer $j \in Y$ that sum to exactly k .
- For each $k \in Z$: check whether $c_k > 0$.

$$m = 19, n = 3$$

$$X = \{ 4, 7, 10 \}$$

$$Y = \{ 5, 8, 15 \}$$

$$Z = \{ 4, 13, 19 \}$$

a yes instance

$$(4 + 15 = 19)$$

$$A(x) = x^4 + x^7 + x^{10}$$

$$B(x) = x^5 + x^8 + x^{15}$$

$$C(x) = x^9 + 2x^{12} + 2x^{15} + x^{18} + x^{19} + x^{22} + x^{25}$$